Markt&Technik

Jürgen Hückstädt

BASIC 7.0 auf dem Commodore 128

Strukturiertes Programmieren erklärt an zahlreichen Anwendungsbeispielen



BASIC 7.0 auf dem Commodore 128

Jürgen Hückstädt

BASIC 7.0
auf dem Commodore 128

Strukturiertes Programmieren erklärt an zahlreichen Anwendungsbeispielen

Markt&Technik Verlag AG

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Hückstädt, Jürgen:
BASIC 7.0 [sieben null] auf dem Commodore 128:
strukturiertes Programmieren, erklärt an zahlr. Anwendungsbeispielen / Jürgen Hückstädt. —
3. Aufl. — Haar bei München : Markt-und-Technik-Verlag, 1986.
ISBN 3-89090-149-2

15 14 13 12 11 10 9 8 7 6 5
90 89 88 87

ISBN 3-89090-149-2

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Schoder, Gersthofen
Printed in Germany

Vorwort

Dieses BASIC-Einführungsbuch ist eine wertvolle Hilfe für jeden, der das leistungsstarke BASIC 7.0 auf dem Commodore 128 PC erlernen möchte. Es spricht sowohl den Computer-Neuling, als auch den Umsteiger vom Commodore 64 oder von einem anderen Computer an, der sich die vielen zusätzlichen BASIC-Anweisungen, die BASIC 7.0 bietet und die anhand zahlreicher Beispiele erläutert werden, zu eigen machen möchte.

Mit BASIC 7.0 hat Commodore erstmals eine BASIC-Version herausgebracht, die sich sehen lassen kann! Sie enthält viele Anweisungen, die man bisher bei Commodore-Computern vergeblich suchte oder nur unter Zuhilfenahme von BASIC-Erweiterungen nutzen konnte. Dies betrifft vor allem die komfortablen Schleifenprogrammierungen mit IF...THEN...ELSE oder DO..LOOP, aber auch Diskettenoperationen, die eine relative Dateiverwaltung einschließen.

Nicht zu übersehen sind auch die vielen Graphik- und Soundanweisungen. Nun ist endlich Schluß mit den seitenlangen DATA-Listings, die mühsam eingepoket werden mußten, wenn man einen Ton aus dem Computer herausbekommen wollte. Darüber hinaus ist es jetzt ein leichtes, Sprites zu definieren und über den Bildschirm zu bewegen. Aber auch das Arbeiten mit hochauflösender und Multicolor-Graphik wird zum Genuß.

Die ersten beiden Kapitel behandeln allgemeine Grundlagen, die ein Anfänger unbedingt durcharbeiten sollte. Dem fortgeschrittenen Programmierer dienen sie zur Wiederholung und Einarbeitung. Die folgenden Kapitel schließen immer mehr die Vorzüge des BASIC 7.0 mit ein. So lernen Sie in Kapitel 3 die komfortable Schleifenprogrammierung mit den neuen BASIC-Anweisungen kennen. Und Kapitel 4 setzt sich mit dem Programmaufbau und der programmierten Fehlerbehandlung auseinander. Kapitel 5 wid-

met sich dann der Feld- und Listenverarbeitung, eine wichtige Grundlage für die eigentliche Dateiverwaltung im 6. Kapitel, die auch relative Dateien einschließt. Schließlich wendet sich das letzte Kapitel an alle Freunde von Graphik und Sound, die die vielen neuen BASIC-Anweisungen auf diesem Gebiet zu schätzen wissen.

Der Autor

Inhaltsverzeichnis

1 Einführung

1.1 BASIC - die Allroundsprache

Dieses Buch behandelt das BASIC 7.0 für den Commodore PC 128. Zunächst wollen wir uns aber Klarheit darüber verschaffen, was BASIC überhaupt ist und einen kurzen Blick in die geschichtliche Entwicklung von Computern werfen.

Viele von uns haben noch mechanische Rechenmaschinen oder den guten alten Rechenschieber in Erinnerung, die einzigen Hilfsmittel, die es früher gab, um Rechenvorgänge schneller, genauer und vor allem bequemer durchzuführen, als es mit Kopfrechnen möglich war. Im Zuge der technologischen Entwicklung suchte man bald nach Möglichkeiten, Berechnungen auch auf dem elektronischen Wege durchführen zu können.

Ende der vierziger Jahre entstanden die ersten Computer, die sich aber noch völlig von den heutigen unterschieden, insbesondere was den inneren Aufbau, die Größe und die Leistungsfähigkeit betraf. Sie arbeiteten noch mit echten Relais, um Stromkreise zu schließen bzw. zu unterbrechen. Ein solches Relais konnte folglich nur eine logische Null-/Eins-Entscheidung übernehmen, die man auch als wahr/falsch oder ja/nein auf der niedrigsten Betriebsebene eines Computers interepretieren kann.

An dieser Stelle wird uns auch klar, daß ein Computer nur auf binärer Ebene arbeiten, d.h. nur zwischen den beiden oben genannten Zuständen unterscheiden kann. Bei der Abarbeitung eines Programms geschieht dies jedoch tausend- und millionenfach. Diese kleinste Speicher- und Informa-

tionseinheit, die nur zwei Zustände bzw. Werte annehmen kann, wird in der Computertechnik als Bit bezeichnet. In der binären Schreibweise bezeichnet man diese Werte als 0 und 1. Wenn wir nun ein zweites Bit dazunehmen, verdoppeln sich die Informationsmöglichkeiten auf insgesamt vier Werte, wie das folgende Beispiel zeigt:

0 = 00   1 = 01   2 = 10   3 = 11

Jetzt fügen wir ein weiteres Bit hinzu, wodurch sich die Informtionsmöglichkeiten auf acht verdoppeln:

0 = 000   1 = 001   2 = 010   3 = 011
4 = 010   5 = 101   6 = 110   7 = 111

Wir sehen also, daß durch Zuschaltung eines jeden Bits oder Schaltkreises doppelt soviele Werte dargestellt werden können. Mathematisch ausgedrückt wachsen sie im Verhältnis zwei hoch der Anzahl der Bits.

Sie können sich sicher denken, wieviele Schaltkreise oder Relais ein Oldtimer-Computer besessen haben muß, um arithmetische Berechnungen mit 10 Stellen Genauigkeit auszuführen. Kein Wunder also, daß man früher einen haushohen Computer brauchte, um die einfachsten Berechnungen durchzuführen zu können. Darüberhinaus waren diese Monstren auch extrem störanfällig und hatten einen enormen Strombedarf.

Im Laufe der Zeit wurden die Relais durch Elektronenröhren und später durch Transistoren ersetzt, wodurch die Computer immer kleiner und leistungsfähiger wurden. Heute, im Zeitalter der Mikroelektronik, können in einem winzigen Chip Tausende solcher Schaltkreise untergebracht werden. Dabei werden meist mehrere Bits zu einer größeren Einheit zusammengefaßt. Viele Computer, wie auch der PC 128, speichern und verarbeiten die Informationen in Einheiten von 8 Bit, die man auch als Bytes bezeichnet. Demnach ist Ihr PC 128 also ein 8-Bit-Rechner. Ein Byte kann demgemäß 2*2*2*2*2*2*2*2 oder 2 hoch 8 bzw. 256 verschiedene Zustände annehmen bzw. Zahlenwerte darstellen.

Stellen Sie sich jetzt einmal vor, Sie müßten Ihren Computer mit Bits und Bytes füttern oder programmieren und sollten auf dieser untersten Maschinenebene die Zahlen 4.8 und 13.5 miteinander multiplizieren. Sicher würden Sie bald die Lust verlieren und das Ergebnis schneller im Kopf

ausrechnen, ganz abgesehen davon, daß Sie fundierte Kenntnisse in der Maschinenprogrammierung benötigten und diese auch fehlerfrei anwenden müßten. Für spezielle Anwendungsgebiete wird diese Programmierungsart zwar noch gelegentlich eingesetzt, aber darauf soll in diesem Buch nicht weiter eingegangen werden.

Schon sehr frühzeitig erkannte man diese Schwierigkeiten und suchte nach Wegen, den Computer auch einfacher programmieren zu können. Im Laufe der Zeit wurden deshalb sogenannte höhere Programmiersprachen entwikkelt. Dabei genügt oft eine einfache und leicht verständliche Anweisung, um hochkomplizierte Vorgänge im Innern des Computers auszulösen. Auf dem PC 128 können Sie Berechnungen wie die oben genannte Multiplikation genauso einfach wie mit einem Taschenrechner durchführen.

Eine dieser Programmiersprachen ist BASIC. Sie wurde in Dartmouth (USA) in den frühen sechziger Jahren entwickelt und galt als leicht zu erlernende Sprache, um Anfängern das Programmieren auf Mikrocomputern beizubringen.

Anfangs war BASIC eine Programmiersprache mit relativ wenigen Anweisungen. Im Laufe der Jahre wurde es jedoch ständig weiterentwickelt und erweitert. Heute ist es die Standard-Sprache der meisten Personal- und Homecomputer, die unproblematisch auf den verschiedensten Gebieten angewandt werden kann. Allerdings gibt es heute eine Reihe von BASIC-Dialekten, die zwar alle auf dem urspünglichen BASIC aufbauen, sich aber dennoch teilweise stark unterscheiden. So haben die meisten Computer ihren eigenen BASIC-Dialekt, der nicht ohne weiteres auch von anderen Computern verstanden werden kann.

Neben BASIC gibt es noch eine Vielzahl anderer Programmiersprachen, die größtenteils auf bestimmte Anwendungsbereiche zugeschnitten sind. FORTRAN, der Vorgänger von BASIC, wurde beispielsweise für den technisch-wissenschaftlichen Bereich und COBOL wurde für kaufmännische Anwendungen entwickelt. FORTH ist eine Sprache mit relativ wenigen Befehlen und wird für die maschinennahe Programmierung angewandt. Eine andere Sprache für strukturiertes Programmieren ist ALGOL, aus der später PASCAL und in jüngster Zeit auch ADA hervorgegangen sind. LOGO ist dagegen eine reine Lernsprache, mit der besonders Kinder den Umgang mit dem Computer einüben können. Weiter Sprachen sind C, COMAL, LISP, PL/1, um nur noch einige Beispiele zu nennen.

BASIC ist sicher deshalb so verbreitet, weil es auch heute noch leicht zu erlernen ist und die Vorzüge anderer Sprachen mehr oder weniger vereinigt, was allerdings von Dialekt zu Dialekt verschieden ist. Bei vielen Computern, wie auch beim Commodore PC 128, ist BASIC direkt nach dem Einschalten verfügbar und muß nicht erst geladen werden. Darüber hinaus kann es sowohl als Interpreter- als auch als Compiler-Sprache Anwendung finden. Ein Interpreter liest den BASIC-Text, wie er eingegeben wurde, und arbeitet ihn direkt ab; ein Compiler dagegen übersetzt ihn erst in Maschinensprache oder in einen maschinennahen Code.

Die meisten der oben genannten Programmiersprachen sind reine Compiler-Sprachen, deren Programme zwar schneller ablaufen, dafür aber nicht so leicht ausgetestet und korrigiert werden können. Nach jeder Änderung müssen sie erneut übersetzt oder compiliert werden.

Das BASIC 7.0 des Commodore PC 128 ist eine reine Interpretersprache, bei der Programme nur eingetippt werden müssen und dann sofort ablauffähig sind. Dies ermöglicht ein äußerst bequemes Programmieren, bei dem Fehler schnell erkannt und behoben werden können.

1.2 Das leistungsstarke BASIC 7.0 des Commodore 128 PC

Falls Sie zu den Lesern gehören, die bereits mit dem Commodore 64 gearbeitet haben, ist Ihnen sicher das BASIC 2.0 bekannt. Der C64 war zwar bereits mit vielen gleichen Eigenschaften wie der PC 128 ausgestattet, jedoch fehlten hier häufig geeignete BASIC-Befehle, um diese auch nutzbar zu machen. Der Anwender mußte entweder auf sie verzichten oder über umfangreiche Kenntnisse des Betriebssystems und der Assemblerprogrammierung verfügen, um das Letzte aus dem Computer herauszulocken.

Gemeint sind hier in erster Linie die Befehle für Graphik und Sound. Unzählige DATA-Zeilen sowie zahlreiche POKE- und PEEK-Befehle waren hierfür nötig und strapazierten die Geduld eines manchen Programmierers. Es gab aber auch viele BASIC-Erweiterungen auf Diskette oder in einem Modul zu kaufen, um diesem Dilemma mehr oder weniger Abhilfe zu schaffen. Ein Beispiel hierfür ist Simons-Basic.

Aber auch die Bedienung der Peripheriegeräte war zwar von BASIC aus möglich, benötigte aber zum Teil recht umfangreiche Befehle. Ein typisches Beispiel hierfür sind einige Diskettenoperationen, wie etwa die Verwaltung von relativen Dateien. Als weiteres gab es kein zeilenbezogenes RESTORE für DATA-Anweisungen, so daß der DATA-Zeiger immer auf die erste DATA-Anweisung zurückgesetzt wurde. Zur Schleifenprogrammierung stand lediglich die FOR...NEXT-Anweisung für eine vorgegebene Anzahl von Wiederholungen zur Verfügung; eine Abbruchbedingung würde erhöhten Programmieraufwand erfordern.

Mit dem leistungsstarken BASIC 7.0 des PC 128 hat Commodore endlich eine BASIC-Version herausgebracht, die sich sehen lassen kann und solchen Einschränkungen nicht mehr unterliegt! Darüber hinaus enthält dieses BASIC sämtliche Befehle der früheren BASIC-Versionen 2.0 (C-64), 3.5 (C-16 und Plus/4) und 4.0 (CBM 80xx). Es stellt echte Grafikbefehle wie DRAW, BOX oder CIRCLE zur Verfügung, unterstützt die Programmierung von Sprites und Sound und ermöglicht die Abfrage von Joystick, Lightpen und Paddles. Damit ist mit unübersichtlichen POKE- und PEEK-Befehlen und den unendlich vielen DATA-Zeilen endlich Schluß.

Aber auch andere Unbequemlichkeiten gehören jetzt der Vergangenheit an. BASIC 7.0 enthält die PRINT USING-Anweisung, welche die formatierte Ausgabe stark vereinfacht. Mit WINDOW können Sie Bildschirmfenster definieren, auf die sich alle PRINT- und INPUTAnweisungen beziehen. Eine Programmunterbrechung kann jetzt durch die SLEEP-Anweisung erreicht werden und benötigt keine FOR...NEXT-Schleife mehr. Für das Diskettenhandling stehen sämtliche komfortablen Befehle des BASIC 4.0 zu Verfügung, die auch die Verarbeitung relativer Dateien mit einschließen. Auch können Sie sich jederzeit das Inhaltsverzeichnis der Diskette ansehen, ohne ein im Arbeitsspeicher befindliches BASIC-Programm zu zerstören.

Verzweigungen und Schleifen lassen sich unter BASIC 7.0 elegant programmieren. IF...THEN...ELSE ist hier ebenso selbstveständlich wie DO WHILE...LOOP bzw. DO UNTIL...LOOP und BEGIN...BEND. EXIT verläßt eine Schleife und setzt die Programmausführung mit der ersten Anweisung hinter der Schleife fort. Somit entfällt die GOTO-Anweisung, zu der man immer die richtige Zeilennummer einsetzen mußte.

BASIC 7.0 ermöglicht eine Fehlerbehandlung ohne Programmabbruch. Die TRAP-Anweisung verzweigt auf die Zeile, in der die Fehlerbehandlung beginnt. Gleichzeitig enthält die Variable ER die Fehlernummer und EL die Nummer der Zeile, in der der Fehler auftrat. Nach Ausführung der Fehlerbehandlung setzt die Anweisung RESUME die normale Programmausführung fort.

Der PC 128 bietet darüber hinaus einige Programmierhilfen wie AUTO (automatische Zeilennumerierung), RENUMBER (Neunumerierung der Programmzeilen einschließlich aller Verzweigungsanweisungen mit Zeilennummern), TRON/TROFF (schrittweise Programmverfolgung) sowie DELETE (Löschen von Programmzeilen).

Nicht zu vergessen ist auch der 80-Zeichen-Bildschirm, der wahlweise eingeschaltet werden kann. Damit wird ein Handicap beseitigt, das eine vernünftige Textverarbeitung bisher nicht möglich machte. Allerdings besitzt der PC 128 Anschlüsse für zwei Bildschirme, einen Composite- und einen RGBI-Monitor. Der CompositeMonitor - notfalls genügt auch ein Fernsehgerät - dient zur Wiedergabe von HIRES-Graphiken (200x320 Punkte) oder 40 Textzeichen pro Zeile, wie beim C-64 üblich. Der RGBI-Monitor liefert dagegen ein schärferes Bild mit einer Auflösung von 200x640 Punkten, die für eine 80-Zeichen-Darstellung unerläßlich ist und deshalb auch zu diesem Zweck auf dem PC 128 Verwendung findet. Die Ausgabe hochauflösender Graphik ist über den RGB-Ausgang allerdings (leider!) nicht möglich.

Im BASIC 7.0 gibt es einen SLOW- und einen FAST-Modus. Im SLOW-Modus wird der 8502-Mikroprozessor mit 1 MHz (etwa wie beim C-64) und im FAST-Modus mit 2 MHz getaktet, was eine doppelt schnelle Abarbeitung der Programme zur Folge hat. Der 80-Zeichen-Schirm steht unter beiden Betriebsarten zur Verfügung.

1.3 Die Hardware - Zusammenstellung des Systems

Bevor wir mit der eigentlichen BASIC-Programmierung beginnen, wollen wir einen kurzen Blick auf den Aufbau der 128 PC-Anlage werfen und uns mit den wichtigsten Ausbaumöglichkeiten beschäftigen.

Wenn Sie sich den Commodore 128 PC bei Ihrem Händler besorgen, erhalten Sie noch lange keine komplette, funktionstüchtige Computeranlage. Was als 128 PC angeboten wird, ist lediglich das Grundgerät, also die Zentraleinheit, die erst im Zusammenhang mit anderen Geräten ihre Arbeit aufnehmen kann.

1.3.1 Grundgerät

Nach dem Auspacken des Computers sehen Sie ein flaches, formschönes und beigefabenes Gehäuse vor sich. Auf den ersten Blick fällt die große und bedienerfreundliche Tastatur ins Auge, die weit mehr Tasten enthält als eine normale Schreibmaschine.



Unter diesem Gehäuse verbirgt sich aber eine große Menge an Intelligenz und enormes Speichervermögen. Als wichtigstes Merkmal sei hier betont, daß Ihr 128 PC eigentlich drei verschiedene Computer in einem enthält: einen altbewährten Commodore 64, einen CP/M-Rechner und, was uns in diesem Buch am meisten interessiert, einen leistungsstarken BASIC-7.0-Rechner mit 140 Befehlen und etwa 122K BASIC-Speicher.

Zusammen mit Ihrem Computer erhalten Sie noch ein Handbuch, ein Netzteil sowie ein Antennenkabel geliefert, das sie benötigen, wenn Sie ein Fernsehgerät an Ihren 128 PC anschließen möchten.

1.3.2 Bildschirm

Apropos Fernsehgerät. Um mit Ihrem 128 PC auch arbeiten zu können, benötigen Sie einen Bildschirm. Als einfachste Möglichkeit bietet sich hierzu ein Schwarzweiß- oder Farbfernsehgerät an. Mit dieser minimalen Grundausstattung können Sie bereits Ihren 128 PC betreiben.

Auf der Rückseite des Computers befindet sich eine kleine runde Buchse, in die Sie das Antennenkabel einstecken. Das andere Ende des Kabels stecken Sie in den Antenneneingang Ihres Fernsehgerätes. Jetzt schließen Sie das Netzteil an. Der Netzstecker kommt in die Steckdose und der quadratische Stecker wird rechts in den Computer eingesteckt. Unmittelbar daneben befindet sich der Ein- und Ausschalter Ihres 128 PC.

Schalten Sie jetzt Ihren Computer und das Fernsehgerät ein und wählen Sie Kanal 36 im UHF-Bereich. Dies ist der Frequenzbereich, auf dem in der Regel das ZDF und die Dritten Programme senden. Haben Sie den richtigen Kanal gefunden, sehen Sie folgende Einschaltmeldung auf dem Bildschirm:

```
      COMMODORE BASIC V7.0 122365 BYTES FREE
       (C)1985 COMMODORE ELECTRONICS, LTD.
             (C)1977 MICROSOFT CORP.
               ALL RIGHTS RESERVED

READY.
```

Jetzt können Sie auf Ihrem 128 PC mit dem Programmieren beginnen!

Sollten Sie Ihren 128 PC nur im C-64-Modus betreiben wollen, geben Sie nach dem Einschalten einfach GO64 ein, und es steht Ihnen ein kompletter

Commodore 64 zur Verfügung. Allerdings beschäftigen wir uns in diesem Buch nicht gesondert mit dem BASIC 2.0, denn es ist einerseits voll im Befehlssatz des BASIC 7.0 enthalten und andererseits ist eine große Anzahl von Büchern auf dem Markt, die sich speziell mit dem Commodore 64 und dessen BASIC-Programmierung befassen.

Nun erzeugt ein Fernsehgerät keinesfalls die optimale Bildqualität, die letztlich in Ihrem Computer steckt. Dies liegt darin begründet, daß das Bild- und Tonsignal, ähnlich wie bei einem Fernsehsender, zunächst mit einem Hochfrequenzträger überlagert (moduliert), der im Empfangsteil des Fernsehgerätes selbst wieder entfernt (demoduliert) wird. Der Hochfrequenzträger hat jedoch eine zu geringe Bandbreite, um das Videosignal in guter Qualität zu übertragen. Deshalb sollte ein Fernsehgerät lediglich als Notbehelf dienen.

Eine weitaus bessere Qualität bietet ein Monitor mit einem Composite-Eingang, der nicht nur ein schärferes Bild, sondern auch reinere Farben erzeugt, falls es sich um einen Farbmonitor handelt. Leider überträgt der Composite-Ausgang des 128 PC nur 40 Zeichen pro Zeile und selbst die hochauflösende Graphik umfaßt nur 320 * 200 Pixels.

Heutzutage haben aber die meisten Personal-Computer einen 80- Zeichen-Bildschirm. Der Commodore 128 PC bietet diese Möglichkeit auch, allerdings nur über einen zweiten Monitor, der der RGBI-Norm angepaßt sein muß. Angeschlossen wird er über die schmale D-Buchse an der Rückseite des Gehäuses. Gegenüber der Composite-Norm erzeugt die RGBI-Norm eine noch bessere Schärfe und Farbtrennung, da die drei Farbkomponenten (Rot, Grün, Blau) separat übertragen werden.

Der RGBI-Ausgang hat zwar eine Auflösung von 80 Zeichen pro Zeile, wird aber nicht von der hochauflösenden Graphik unterstützt. Somit ist der Anwender, was die Graphik anbetrifft, auf die Auflösung angewiesen, die bereits der C-64 bietet.

Um überhaupt in den 80-Zeichen-Modus zu gelangen, muß noch vor dem Einschalten des Gerätes die 40/80-DISPLAY-Taste eingerastet werden. Dann nämlich arbeitet der RGBI-Ausgang im 80-Zeichen-Modus, während der Composite-Ausgang lediglich für Graphik reserviert ist. Zur vollen Ausnutzung der Fähigkeiten des PC 128 sind also zwei verschiedene Monitore erforderlich. Commodore bietet den Monitor 1902 an, der wahlweise nach der Composite- und nach der RGBI-Norm betrieben werden kann.

Wir wissen bereits, daß der 128 PC einen SLOW- und einen FAST-Modus besitzt. Im SLOW-Modus arbeitet er etwa mit der gleichen Geschwindigkeit wie der C-64, nämlich mit 1 MHz, während er im FAST-Modus mit 2 MHz getaktet wird. In diesem Modus kann jedoch nur der RGBI-Monitor mit 40- oder 80-Zeichen-Darstellung betrieben werden, wobei es allerdings keine Graphikmöglichkeiten gibt.

1.3.3 Datasette

Nachdem wir nun das Problem mit dem Bildschirm gelöst haben, müssen wir uns überlegen, welches Speichermedium wir verwenden. Die einfachste und billigste Möglichkeit bietet die Datasette. Sie ist im Prinzip ein umgebauter, handelsüblicher Kassettenrekorder und wird auch mit normalen Tonkassetten betrieben. Die Nachteile sind eine sehr geringe Aufzeichnungs- und Wiedergabegeschwindigkeit und die große Gefahr, daß bei mehreren Dateien auf einem Band Überschneidungen auftreten können, wenn man sich nicht genau anhand des Zählwerkes Bandstelle und Länge merkt. Auch das Arbeiten mit Direktzugriffs- oder relativen Dateien ist nicht möglich.

1.3.4 Floppylaufwerk

Eine wesentlich komfortablere Möglichkeit zum Speichern von Programmen und Daten bietet ein Floppylaufwerk, das mit Disketten als Datenträger arbeitet. Eine Diskette ist eine in einer Schutzhülle eingeschlossene runde Magnetscheibe, die sich im Laufwerk mit hoher Geschwindigkeit dreht, während Daten auf sie geschrieben bzw. von ihr gelesen werden.

Das Aufzeichnungsverfahren einer Diskette unterscheidet sich von dem einer Kassette ganz wesentlich. Während der Kassettenrekorder die Daten hintereinander (sequentiell) aufnimmt, werden sie auf der Diskette in einzelnen Sektoren von 256 Bytes abgelegt. Damit dies möglch wird, muß eine fabrikneue Diskette zunächst einmal formatiert werden. Bei der Formatierung werden Spuren erzeugt, die konzentrische Kreise auf der Diskette bilden. Jede Spur ist wiederum in eine Anzahl von Sektoren unterteilt, von denen bei den Commodore-Laufwerken jeder 256 Bytes an Informationen aufnehmen kann.

Wird nun ein Programm oder eine Datei auf Diskette geschrieben, so wird sie, prinzipiell betrachtet, in 256-Byte-Abschnitten verschiedenen Sektoren

zugeordnet. Jeder Sektor enthält außerdem einen Zeiger, der auf den jeweils nächsten Sektor der Datei zeigt. Außerdem werden alle belegten Sektoren in einem besonderen Verzeichnis vermerkt. Beim Schreiben einer neuen Datei gibt dieses Belegungsverzeichnis darüber Auskunft, welche Sektoren bereits belegt und welche noch frei sind. In diesem Zusammenhang wird die Datei auch in dem Inhaltsverzeichnis (Directory) eingetragen, das neben ihrem Namen noch Informationen über den Dateityp, die Anzahl der belegten Sektoren und - für den Benutzer nicht sichtbar - einen Zeiger auf den ersten Sektor der Datei enthält.

Darüber hinaus bietet ein Floppylaufwerk die Möglichkeit, auf einzelne Sektoren direkt zuzugreifen oder relative Dateien anzulegen, die im Prinzip ebenfalls ein Direktzugriffsverfahren darstellen. Besitzt man eine Datei mit vielen Datensätzen, muß nicht immer erst die gesamte Datei eingelesen und durchsucht werden, da auf jeden Datensatz direkt zugegriffen wird. Deshalb läßt sich mit Direktzugriffsdateien auch wesentlich schneller arbeiten als mit sequentiellen Dateien. Das Arbeiten mit solchen Dateien werden wir im Laufe dieses Buches noch genauer kennenlernen.

Zusammen mit dem 128 PC hat Commodore auch ein dazu passendes Floppylaufwerk mit der Bezeichnung 1571 auf den Markt gebracht. Dieses Laufwerk wird an den seriellen Bus (serielle Schnittstelle) des 128 PC angeschlossen. Im C-128-Modus erfolgt ein fünf- bis sechsfach schnellerer Datentransfer als beim Vorgängermodell 1541. Darüber hinaus werden hier die Disketten beidseitig beschrieben, wodurch eine doppelte Aufzeichnungskapazität erreicht wird.

Sollten Sie bereits im Besitz einer 1541-Floppy sein, können Sie diese jedoch mit dem 128 PC betreiben; dabei geht allerdings der Geschwindigkeitsvorteil und die doppelseitige Aufzeichnung verloren. Disketten, die mit der 1571-Floppy im C-128-Modus beschrieben wurden, sollten Sie, um die Gefahr des Datenverlustet zu vermeiden, nicht mit der 1541-Floppy lesen und umgekehrt. Ist der C-64-Modus eingeschaltet, arbeitet die Floppy 1571 genauso wie die Floppy 1541.

Hier noch ein Hinweis für alle Leser, die bereits mit dem C-64 gearbeitet haben. Reine BASIC-Programme, die mit dem BASIC 2.0 geschrieben wurden, laufen auch auf dem 128 PC unter BASIC 7.0, sofern sie keine Zugriffsbefehle auf den Speicher oder das Betriebssystem enthalten. Achten Sie deshalb darauf, daß sie frei von POKE-, PEEK-, SYS- und USR-Befehlen sind. Solche BASIC-Programme können Sie über die Floppy 1541 oder über Kassette laden und dann unter BASIC 7.0 ausführen.

Für Ihr Floppylaufwerk sollten Sie nur Marken-Disketten mit guter Qualität verwenden. So manches Billigangebot in Fachzeitschriften entspricht nicht dem erforderlichen Quälitätsstandard. Geben Sie deshalb lieber ein paar Mark mehr für Ihre Disketten aus, um die Wahrscheinlichkeit von Datenverlusten auf ein Minimum zu reduzieren. Auf keinen Fall sollten Sie es jedoch versäumen, von allen wichtigen Programmen und Dateien eine Sicherheitskopie anzulegen, möglichst auf einer anderen Diskette.

1.3.5 Drucker

An den seriellen Bus können Sie auch einen Drucker anschließen. In Frage kommen hier in erster Linie die Commodore-Drucker MPS 801 und MPS 802 (vormals 1526). Beide besitzen den kompletten Commodore-Zeichensatz und sind graphikfähig, d.h. die Drucknadeln können einzeln angesprochen werden. Allerdings unterscheiden sich beide Drucker bei der Programmierung der Einzelnadelansteuerung. Soll eine Hardcopy vom Bildschirm erstellt werden, empfiehlt es sich ohnehin aus Zeitgründen, dies mit Hilfe eines Maschinenprogramms vorzunehmen, auf das wir aber im Rahmen dieses BASIC-Lehrbuches nicht näher eingehen wollen.

Mit Sicherheit wird, ähnlich wie für den C-64, bald auch ein Centronics-Interface (parallele Schnittstelle) für den 128 PC angeboten. Mit einem solchen Interface können Sie jeden Drucker betreiben, der mit einer Centronics-Schnittstelle ausgestattet ist (z.B. EPSON-Drucker). Fast alle diese Drucker verarbeiten aber den Standard-ASCII-Zeichensatz, und meistens ist auch der deutsche Zeichensatz einstellbar. Viele Interfaces nehmen eine Codewandlung bereits automatisch vor; ist dies nicht der Fall, müssen Sie sich ein eigenes Treiberprogramm schreiben.

Ein weiterer Vorteil vieler markenfremder Drucker mit Centronics-Schnittstelle ist die Fähigkeit, verschiedene Schrifttypen, Schriftbreiten, Unterstreichungen, Unterlängen etc. zu drucken. Dazu sind allerdings spezielle Steuerzeichen nötig, die leider nicht genormt sind. Deshalb ist auf jeden Fall eine Anpassung nötig.

Über die Centronics-Schnittstelle können Sie auch verschiedene Typenraddrucker anschließen. Solche Drucker besitzen zur Erzeugung von Zeichen keine Drucknadeln, wie es bei den bisher betrachteten Matrixdruckern der Fall war, sondern - wie der Name bereits besagt - ein Typenrad.

Für den Hausgebrauch ist ein Matrixdrucker meist vollkommen ausreichend. Nur für besondere Zwecke sollte man sich einen Typenraddrucker zulegen, der überdies auch wesentlich teurer ist. Falls Sie lediglich Ihre Programmlistings und einige Tabellen ausdrucken wollen, genügen die Commodore Drucker MPS 801 und MPS 802 vollauf. Sie bewegen sich preislich bei ca. 800 DM und benötigen kein Centronics-Interface, das weitere 300 DM kostet.

1.3.6 Noch einmal - die Tastatur

Der Commodore 128 PC hat gegenüber seinem Vorgängermodell C-64 eine wesentlich erweiterte Tastatur, wie sie ansonsten nur bei Personal-Computern anzutreffen ist. Besonders auffällig ist die gesonderte Zifferntastatur rechts neben der Haupttastatur. Damit lassen sich recht einfach große Zahlenkolonnen in den Rechner eingeben, was speziell im kaufmännischen Bereich oder bei der Tabellenkalkulation sehr nützlich und hilfreich ist. Auch Leser, die es gewohnt sind, Maschinenprogramme in Form von endlosen DATA-Anweisungen einzugeben, werden diese Tastatur zu schätzen wissen. Außer den Ziffern 0 bis 9 enthält die Zifferntastatur noch eine Plus-, Minus- und Punkttaste. Nicht zu übersehen ist die hochstehende ENTER-Taste, die die gleiche Funktion wie die RETURN-Taste auf der Haupttastatur besitzt. Wahrscheinlich wurde sie nur deshalb an dieser Stelle angebracht, um ein ebenso komfortables Arbeiten wie mit einer Registriermaschine zu ermöglichen.

Die eigentliche Haupttastatur ist im Aufbau mit der des Commodore-64 völlig identisch. So ist auch zu erklären, daß im C-64-Modus nur diese Tastatur und die vier Funktionstasten funktionieren, während alle anderen Tasten abgeschaltet sind.

Die Funktionstasten sind oberhalb der Zifferntastatur angebracht und zwar waagerecht, im Gegensatz zu der senkrechten Anordnung beim C-64. Sie sind hier bereits vorbelegt, wie die folgende Aufstellung zeigt.

Taste	BASIC-Anweisung	Funktion
F1	GRAPHIC	Umschaltung in den Graphik-Modus
F2	DLOAD"	BASIC-Pogramm laden
F3	DIRECTORY	Anzeige des Disketten-Inhaltsverzeichnisses
F4	SCNCLR	Löscht den Bildschirm (nur Graphik-Modus)
F5	DSAVE"	BASIC-Programm speichern
F6	RUN	Lädt BASIC-Programm und startet es, falls mit Namen
F7	LIST	Listet BASIC-Programm auf dem Bildschirm
F8	MONITOR	Schaltet den Maschinen-sprachemonitor ein

Diese Vorbelegung mit Funktionen ist eine große Erleichterung für den Anwender, der häufig mit ihnen arbeiten muß. Man kann die Belegung aber auch ändern und sie seinen eigenen Wünschen anpassen. Dies geschieht mit der KEY-Anweisung.

Angenommen, Sie möchten die Belegung der Funktionstaste F4 so abändern, daß nicht mehr der Graphik- sondern der normale Textbildschirm gelöscht wird. Geben Sie dazu folgendes ein:

```
KEY 4, CHR$(147)+CHR$(13)
```

oder

```
KEY 4, "<CLR>"+CHR$(13)
```

Beide Anweisungen sind völlig identisch. Um den Bildschirm zu löschen, können Sie entweder

```
PRINT CHR$(147)    oder    PRINT "<CLR>"
```

eingeben. <CLR> steht hier für das Drücken der CLR-Taste rechts oben auf der Tastatur (mit SHIFT). CHR$(13) ist gleichbedeutend mit dem Drücken der RETURN-Taste, denn wir wollen ja nicht nur die Anweisung auf den Bildschirm schreiben, sondern sie auch gleich ausführen.

Links neben den Funktionstasten befinden sich vier Cursor-Tasten, deren Funktion mit denen rechts unten völlig identisch ist.

Nachfolgend nun noch kurz die Beschreibung der restlichen Sondertasten in der oberen Reihe:

Die ESC-Taste findet im Zusammenhang mit einer oder mehreren Texttasten Verwendung, um eine Escape-Code-Sequenz unter CP/M zu erzeugen.

TAB ist eine Tabulatortaste, mit der der Cursor an die nächste Tabulatorposition gesetzt wird.

Mit der ALT-Taste können beliebigen Tasten Funktionen zugeordnet werden, ähnlich wie es bei den Funktionstasten der Fall ist. Soll eine Funktion aufgerufen werden, ist die zugeordnete Taste zusammen mit der ALT-Taste zu drücken.

Wird die ASCII-DIN-Taste im 40-Zeichen-Modus gedrückt, schaltet der 128 PC auf den deutschen Zeichensatz um. Dabei wird gleichzeitig die Tastatur einer deutschen DIN-Tastatur angepaßt, d.h. einige Tasten haben nicht mehr ihre ursprüngliche Bedeutung.

Die HELP-Taste dient zum leichteren Aufsuchen von Fehlern in BASIC-Programmen. Sie sollte immer dann gedrückt werden, wenn das Programm aufgrund einer Fehlermeldung abbricht. Dadurch wird die Zeile, in der der Fehler auftrat, gelistet und die fehlerhafte Anweisung invers dargestellt.

LINE FEED erzeugt auf dem Bildschirm einen Zeilenvorschub ohne Wagenrücklauf (Carriage Return). Wird diese Taste gedrückt, wandert der Cursor um eine Zeile nach unten, ohne seine derzeitige Spaltenposition zu verlassen.

40/80 DISP schaltet vom 40 auf den 80-Zeichen-Bildschirm um. Achtung! Diese Taste sollte bereits vor dem Einschalten des Computers ein- bzw. ausgerastet werden. Ansonsten sind nur auf dem RGBI-Monitor 80 Zeichen pro Zeile darstellbar.

NO SCROLL verhindert ein Abwandern des Cursors unter die 25. Bildschirmzeile, unterbricht LIST-Vorgang.

1.4 Direkt- und Programmodus

Ihren Computer können Sie auf zwei Arten betreiben, nämlich im Direkt- und im Programmodus. Im Direktmodus erteilen Sie dem Computer Instruktionen, die er unmittelbar nach der Eingabe abarbeitet. Sie tippen lediglich die entsprechenden Anweisungen ein und drücken die RETURN-Taste.

Anders ist es im Programmodus. Hier müssen Sie zunächst Programmzeilen eingeben, die der Computer in seinem Arbeitsspeicher ablegt. Das Programm wird erst abgearbeitet, wenn der Computer den entsprechenden Befehl dazu erhält.

Im Gegensatz zu vielen anderen Computern besitzt der PC 128 einen komfortablen Bildschirmeditor, dessen Vorzüge wir später noch kennenlernen werden. Ein Editor ist ein internes Maschinenprogramm, das dazu dient, BASIC-Zeilen und sonstige Instruktionen über Tastatur und Bildschirm einzugeben bzw. zu korrigieren (editieren). Ist eine Zeile fertig eingegeben, wird sie durch Drücken der RETURN-Taste abgeschlossen. Im Direktmodus wird sie sogleich ausgeführt und im Programmodus als Programmzeile gespeichert.

Wie versteht nun der Computer, ob er im Direkt- oder Programmodus arbeiten soll? Dies ist ganz einfach: Steht nämlich nach dem Drücken der RETURN-Taste am Zeilenanfang eine Zahl, so handelt es sich um eine Programmzeile, anderenfalls um eine Zeile, die im Direktmodus abgearbeitet wird. Die meisten Anweisungen können sowieso sowohl im Programm- als auch im Direktmodus erteilt werden.

Zum besseren Verständnis wollen wir nun ein paar Beispiele betrachten. Wir kennen bereits den Cursor, das blinkende Kästchen auf dem Bildschirm. Der Cursor erscheint immer dann, wenn der Computer bereit ist, neue Anweisungen über die Tastatur entgegenzunehmen; er zeigt dann auf die Stelle, an der das nächste Zeichen auf dem Bildschirm erscheint. Geben wir ein Zeichen ein, erscheint es an der Cursorposition und der Cursor selbst rückt um eine Stelle nach rechts, wo er auf das nächste Zeichen wartet.

Achten Sie jetzt bitte darauf, daß der Cursor am Anfang einer leeren Bildschirmzeile steht. Wenn nicht, drücken Sie gleichzeitig die SHIFT- und die CLR-/HOME-Taste, was ein Löschen des Bildschirms zur Folge hat. Falls dies in Ausnahmefällen einmal nicht funktioniert und stattdessen merkwürdige reverse Zeichen auf dem Bildschirm erscheinen, drücken Sie einfach die RETURN-Taste und wiederholen diese Anweisung. Unter Umständen erscheint nach dem Drücken der RETURN-Taste die Meldung ?SYNTAX ERROR, die uns aber im Augenblick nicht weiter zu interessieren braucht. Fahren Sie mit den nachfolgend beschriebenen Anweisungen fort.

Zunächst wollen wir im Direktmodus arbeiten; geben Sie dazu folgendes ein:

PRINT "GUTEN TAG, ICH BIN DER PC 128" <RETURN>

Das Wort <RETURN> tippen Sie nicht in Buchstaben ein, sondern drücken statt dessen die RETURN-Taste. Diese Schreibweise wird vorläufig noch beibehalten, um Sie darauf aufmerksam zu machen, daß die Zeile durch Drücken der RETURN-Taste abgeschlossen werden muß.

Haben Sie die Zeile korrekt eingegeben, erscheint auf dem Bildschirm die Meldung:

GUTEN TAG, ICH BIN DER PC 128

READY.

Der Cursor ist wieder sichtbar und signalisiert, daß der Computer bereit ist, weitere Anweisungen entgegenzunehmen.

Wir haben hier bereits den ersten Befehl, PRINT, kennengelernt, der den Computer anweist, eine Zeichenkette auf dem Bildschirm auszugeben. PRINT bezeichnet man auch als Schlüsselwort, das der Computer als Befehl interpretiert. Hinter PRINT steht dann in Anführungszeichen die Zeichenkette, die ausgegeben werden soll.

Versuchen Sie jetzt einmal, die gleiche Anweisung nochmals einzugeben, an den Zeilenanfang aber die Zahl 10 zu setzen:

10 PRINT "GUTEN TAG, ICH BIN DER PC 128" <RETURN>

Nach Drücken der RETURN-Taste erscheint dieses Mal keine Meldung sondern nur der Cursor. Wir haben nämlich die Anweisung, eine Meldung auszugeben, in eine Programmzeile mit der Nummer 10 gepackt. Diese Programmzeile legt der Computer im Speicher ab. Damit erstellten wir ein BASIC-Programm, das aus einer einzigen Zeile besteht und das wir jetzt ablaufen lassen, indem wir

RUN <RETURN>

eingeben. Wieder erscheint die Meldung

GUTEN TAG, ICH BIN DER PC 128

READY.

Im allgemeinen besteht ein BASIC-Programm natürlich aus einer Vielzahl von Programmzeilen, die in aufsteigender Reihenfolge numeriert sind. Dabei sind Zeilennummern zwischen 0 und 63999 zulässig. Zwar gibt es kein BASIC-Programm, das 64000 Zeilen umfaßt, denn dazu wäre der Speicher viel zu klein. Wir können aber verschiedene Programmsegmente mit runden Zeilennummern (z.B. 1000, 2000, 3000 usw.) beginnen lassen, um die Übersichtlichkeit zu erhöhen.

Doch zunächst wollen wir ein kleineres Programm mit mehreren Zeilen schreiben. Geben Sie dazu als nächstes ein:

NEW <RETURN>

Dieser Befehl löscht den BASIC-Arbeitsspeicher, so daß Sie jetzt ein neues Programm eingeben können, das dieses Mal aus drei Programmzeilen bestehen soll. Von nun an verzichten wir in unseren Anweisungen auf den Hinweis <RETURN>, denn wir wissen ja bereits, daß wir jede BASIC-Zeile sowohl im Direkt- als auch im Programmodus durch Drücken der RETURN-Taste abschließen müssen.

```
10 PRINT "GUTEN TAG, ICH BIN DER PC 128"
20 PRINT "UND BRINGE IHNEN GROSSE NEUIGKEITEN,"
30 PRINT "DENN MEIN BASIC 7.0 IST EINFACH SPITZE!!!"
```

Geben Sie jetzt wieder

RUN

ein. Darauf läuft das Programm ab und gibt folgende Meldung(en) aus:

GUTEN TAG, ICH BIN DER PC 128
UND BRINGE IHNEN GROSSE NEUIGKEITEN,
DENN MEIN BASIC 7.0 IST EINFACH SPITZE!!!

An diesem Beispiel sehen wir ganz deutlich, daß der Computer die Programmzeilen der Reihe nach abarbeitet. Wir haben hier die Zeilennummern 10, 20 und 30 benutzt, was zwar üblich ist, aber keinesfalls so sein muß. Wir hätten ebenso die Nummern 1,2,3 oder 967, 22345, 61120 verwenden können und damit das gleiche Ergebnis erzielt. Versuchen Sie es doch einmal!

Die meisten Programmierer verwenden jedoch Zehnerschritte, da sie dann immer noch die Möglichkeit haben, zwischen zwei Zeilen bis zu neun weitere Zeilen einzufügen. Wie das funktioniert, werden wir gleich sehen.

1.5 Programmierhilfen

Betrachten wir nochmals unser kleines Programm, das wir soeben eingegeben und ausgeführt haben. Wurde es nicht durch NEW gelöscht, steht es immer noch im Speicher und wir könnten es beliebig oft ablaufen lassen, erweitern, ändern, auf Diskette oder Kassette abspeichern usw.

Ist Ihnen eigentlich bei unserem einzeiligen und dreizeiligen Programm etwas aufgefallen? Wenn ja, dann gehören Sie schon zu den fortgeschritteneren Programmierern.

Zeile 10 (unser erstes Programm) ist nämlich auch in dem zweiten Programm exakt enthalten. Daher wäre es nicht unbedingt notwendig gewesen, den Arbeitsspeicher mit NEW zu löschen und dann die gleiche Zeile erneut einzugeben. Wir hätten nur die Zeilen 20 und 30 anhängen müssen, indem

wir sie eintippen und die RETURN-Taste drücken. Dabei spielt es keine Rolle, ob Sie nach Zeile 10 zuerst Zeile 30 und dann Zeile 20 eingeben oder umgekehrt. Der Editor setzt jede eingegebene Zeile an ihre richtige Stelle. Probieren Sie es doch einmal!

Zum Glück müssen wir uns bei der Frage, welche Zeile wir eingegeben haben, nicht auf unser Gedächtnis verlassen. Mit dem LIST-Befehl wird das Programm auf dem Bildschirm ausgegeben. Geben Sie jetzt LIST ein und drücken Sie die RETURN-Taste:

```
LIST
10 PRINT "GUTEN TAG, ICH BIN DER PC 128"
20 PRINT "UND BRINGE IHNEN GROSSE NEUIGKEITEN,"
30 PRINT "DENN MEIN BASIC 7.0 IST EINFACH SPITZE!!!"

READY.
```

Diese drei Zeilen können wir noch bequem auf dem Bildschirm darstellen. Was geschieht aber, wenn wir ein langes Programm listen wollen?

Für den LIST-Befehl gibt es einige Zusatzangaben, die dafür sorgen, daß nur ein Teil des Programms gelistet wird. Versuchen Sie folgendes:

```
LIST 10
10 PRINT "GUTEN TAG, ICH BIN DER PC 128"

READY.
```

Steht hinter LIST eine Zeilennummer, so wird nur die betreffende Zeile ausgegeben. Im folgenden Beispiel werden alle Zeilen zwischen 10 und 20 gelistet:

```
LIST 10-20
10 PRINT "GUTEN TAG, ICH BIN DER PC 128"
20 PRINT "UND BRINGE IHNEN GROSSE NEUIGKEITEN,"

READY.
```

Die Anweisung

```
LIST -20
```

hätte in diesem Fall die gleiche Wirkung, denn sie listet alle Programmzeilen mit den Nummern bis einschließlich 20. Andererseits besteht auch die Möglichkeit, von einer Zeilennummer bis zum Programmende zu listen, wie im folgenden Beispiel:

```
LIST 20-
20 PRINT "UND BRINGE IHNEN GROSSE NEUIGKEITEN,"
30 PRINT "DENN MEIN BASIC 7.0 IST EINFACH SPITZE!!!"

READY.
```

Falls Sie einen Drucker besitzen, können Sie das Listing auch ausdrucken. Schalten Sie den Drucker ein und geben Sie folgende Anweisungen ein, die jeweils mit RETURN abzuschließen sind:

```
OPEN 1,4
CMD 1
LIST
PRINT#1
CLOSE 1
```

Ein Verständnis der einzelnen Anweisungen ist an dieser Stelle noch nicht notwendig, denn die Bedienung der Peripheriegeräte wie Drucker, Floppy etc. wird erst an späterer Stelle behandelt. Für diejenigen unter Ihnen, die sich darin bereits auskennen, hier eine Kurzbeschreibung:

OPEN 1,4 öffnet den Ausgabekanal zum Drucker (Geräteadresse 4) über die logische Filenummer 1, CMD 1 leitet die Bildschirmausgabe auf den Drucker, um und LIST listet das Programm, wobei auch die obengenannten Zusatzangaben über einzelne Programmteile enthalten sein können. PRINT#1 dirigiert die Ausgabe wieder auf den Bildschirm und CLOSE 1 schließt das logische File 1.

BASIC-Versionen anderer Computer haben für diese Prozedur den einfachen Befehl LLIST, der aber im BASIC 7.0, wie auch schon in den früheren BASIC-Versionen von Commodore, leider nicht enthalten ist.

Jetzt wollen wir unser Programm durch Hinzufügen der Zeilen 15 und 40 noch etwas erweitern. Die Reihenfolge wurde dabei absichtlich vertauscht.

```
40 PRINT "DAS ZEIGT IHNEN DIESES MARKT&TECHNIK-
BUCH."
15 PRINT "-DER NEUESTE COMPUTER VON COMMODORE-"
```

Wenn Sie jetzt LIST eingeben, sind die Programmzeilen in der richtigen Reihenfolge aufgeführt, in der sie auch abgearbeitet werden:

```
10 PRINT "GUTEN TAG, ICH BIN DER PC 128"
15 PRINT "-DER NEUESTE COMPUTER VON COMMODORE-"
20 PRINT "UND BRINGE IHNEN GROSSE NEUIGKEITEN,"
30 PRINT "DENN MEIN BASIC 7.0 IST EINFACH SPITZE!!!"
40 PRINT "DAS ZEIGT IHNEN DIESES MARKT&TECHNIK-
          BUCH."
```

Auf die Ausführung dieses Programms wollen wir hier verzichten, da wir ja wissen, in welcher Reihenfolge der Text erscheint. Das einzige, was am optischen Erscheinungsbild stört, ist die Zeile 15. Mit dem RENUMBER-Befehl werden die Zeilen in Zehnerschritten neu durchnumeriert. Geben Sie jetzt

```
RENUMBER
```

ein und listen Sie das Programm nochmals:

```
10 PRINT "GUTEN TAG, ICH BIN DER PC 128"
20 PRINT "-DER NEUESTE COMPUTER VON COMMODORE-"
30 PRINT "UND BRINGE IHNEN GROSSE NEUIGKEITEN,"
40 PRINT "DENN MEIN BASIC 7.0 IST EINFACH SPITZE!!!"
50 PRINT "DAS ZEIGT IHNEN DIESES MARKT&TECHNIK-
          BUCH."
```

RENUMBER ohne Zusatz numeriert das Programm also in Zehnerschritten, beginnend mit Zeilennummer 10, durch. Soll z.B. die erste Zeilennummer nicht 10, sondern 100 lauten, geben Sie

RENUMBER 100

ein. Wünschen Sie darüber hinaus eine Schrittweite von 20, lautet die Anweisung

RENUMBER 100,20

Wenn wir eine neue Programmzeile mit einer Nummer eingeben, die bereits existiert, ersetzt die neue Zeile die alte. Geben Sie dagegen nur eine Zeilennummer ein und drücken RETURN, wird die betreffende Zeile im Programm gelöscht, falls sie bereits existiert.

Wenn wir z.B. in unserem neu durchnumerierten Programm die Zeilen 20 bis 40 löschen möchten, geben wir

20 <RETURN>
30 <RETURN>
40 <RETURN>

ein. Mit der DELETE-Anweisung können wir uns die Arbeit etwas erleichtern, da sie es ermöglicht, mehrere Zeilen gleichzeitig zu löschen. Wenn Sie jetzt eingeben

DELETE 20-40

werden die Zeilen 20 bis 40 gleichzeitig gelöscht.

Die Anweisung

DELETE 30-

löscht alle Zeilen von Nummer 30 bis Programmende und die Anweisung

DELETE -40

löscht alle Zeilen vom Programmanfang bis einschließlich Zeile 40.

Eine weitere Programmierhilfe ist die automatische Zeilennumerierung, die mit der AUTO-Anweisung eingeschaltet wird. Die Anweisung

AUTO 10

schaltet die Schrittweite, die beliebig groß gewählt werden kann, auf 10. Die erste Zeile müssen Sie daraufhin normals mit Zeilennummer eingeben. Nach Drücken der RETURN-Taste erscheint die nächste Zeilennummer. Der Cursor steht bereits an der richtigen Stelle, so daß Sie nur noch den Programmtext eingeben und wieder RETURN drücken müssen usw. Wurde das Programm fertig eingegeben, geben Sie

AUTO

(ohne Zusatz) ein, und die automatische Zeilennumerierung ist wieder abgeschaltet.

1.6 Fehlerkorrektur

Wahrscheinlich fragen Sie sich jetzt, was zu tun ist, wenn Sie beim Eintippen einen Fehler gemacht haben. Eine Möglichkeit besteht darin, die fehlerhafte Zeile neu einzugeben und zu überschreiben. Dies ist allerdings eine sehr umständliche Methode, besonders wenn es darum geht, nur ein Zeichen auszubessern.

Zunächst einmal holen Sie sich mit LIST (Zeilennummer) die betreffende Zeile auf den Bildschirm. Und hier beginnt nun das eingangs erwähnte bequeme Editieren, das längst nicht bei allen Computeren üblich ist.

In der obersten Tastaturreihe des PC 128 befinden sich vier Tasten mit Pfeilen, von denen jeder in eine andere Richtung zeigt. Die Pfeilrichtung entspricht der Bewegung des Cursors auf dem Bildschirm, weshalb diese Tasten auch Cursorsteuertasten heißen. Mit Hilfe dieser Tasten fahren Sie nun den Cursor an die fehlerhafte Stelle, korrigieren diese und drücken anschließend die RETURN-Taste. Das ist alles!

Hier nun ein paar Beispiele. Angenommen, die fehlerhafte Zeile lautet

```
10 PRINT "CONPUTER"
```

Um das N durch ein M zu ersetzen, überschreiben Sie lediglich das falsche Zeichen und drücken anschließend RETURN. Anders ist es im folgenden Fall:

```
10 PRINT "COPUTER"
```

Hier fehlt das M und muß nachträglich eingesetzt werden. Dazu setzen wir den Cursor auf das P und drücken die INS-/DEL-Taste gleichzeitig mit SHIFT. Das P und alle nachfolgenden Zeichen werden um eine Stelle nach rechts verschoben und der Cursor zeigt auf die dadurch geschaffene freie Stelle. Hier setzen wir dann das M ein und drücken RETURN. Bei

```
10 PRINT "COMPUUUUTER"
```

liegt der Fall genau umgekehrt. Wir plazieren den Cursor auf dem T, drücken dreimal die DEL-Taste und schließlich RETURN.

Trotz seiner Einfachheit hat das Editieren noch einen kleinen Haken. Haben Sie nämlich einmal ein Anführungszeichen gesetzt und arbeiten dann mit den Editiertasten, erscheinen solange eigenartige inverse Zeichen auf dem Bildschirm, bis Sie ein weiteres Mal ein Anführungszeichen setzen. In einem solchen Fall drücken Sie einfach die RETURN-Taste und beginnen mit der Korrektur von vorne.

Übrigens haben die eben genannten reversen Zeichen durchaus einen Sinn; sie dienen nämlich der programmgesteuerten Cursorpositionierung. Soll der Cursor z.B. um drei Zeilen nach oben wandern, geben Sie in einem Programm

```
PRINT "<3 x CRSR UP>"
```

ein. "<3 x CRSR UP>" bedeutet hier, daß hinter dem Anführungszeichen dreimal die Cursortaste mit dem Pfeil nach oben zu drücken und anschliessend ein weiteres Anführungszeichen zu setzen ist. Diese Schreibweise ist allerdings ein Notbehelf, dessen wir uns in diesem Buch bedienen, da die reversen Zeichen nur auf dem Bildschirm, nicht aber im Buchdruck zur Verfügung stehen. Geben Sie jedesmal, wenn Sie in diesem Buch eine solche Anweisung finden, die entsprechenden Tasten ein.

Die bisher betrachteten Fehler bezogen sich immer nur auf den Text, der ausgegeben werden sollte. Dabei interessiert es den Computer nicht, ob er richtig oder falsch geschrieben ist. Anders ist es dagegen bei Angaben, die der Computer nicht als Befehl interpretieren kann, wie etwa den Schlüsselwörtern. Hier ein Beispiel:

```
PRIT "ICH BIN DER PC 128"

?SYNTAX ERROR
READY.
```

Der Computer kennt nur das Wort PRINT als Befehl, nicht aber das Wort PRIT und meldet deshalb einen SYNTAX ERROR. Diese Fehlermeldung tritt also immer dann auf, wenn der Computer einen Befehl nicht klar erkennt. Dies gilt sowohl für den Programm- als auch für den Direktmodus. Im Programmodus erscheint zusätzlich noch die Nummer der Zeile, in der der Fehler auftrat, so daß dieser leicht aufzufinden und zu korrigieren ist.

1.7 Programme sichern und laden

Haben wir im Arbeitsspeicher ein BASIC-Programm stehen und schalten den Computer aus, so geht das Programm verloren. Um dies zu verhindern, können wir es auf Kassette oder Diskette abspeichern und bei Bedarf wieder laden.

Beginnen wir mit der Kassette. Schließen Sie Ihre Datasette an und legen Sie eine leere Kassette ein, um ein Programm unter dem Namen "TEST" abzuspeichern. Geben Sie nun

```
SAVE "TEST"
```

ein. Daraufhin erscheint die Meldung

```
PRESS PLAY & RECORD ON TAPE
```

auf dem Bildschirm. Wenn Sie jetzt die RECORD- und PLAY-Taste am Recorder gleichzeitig betätigen, beginnt das Band zu laufen. Erscheint

READY, ist der Speichervorgang abgeschlossen. Drücken Sie jetzt die STOP-/EJECT-Taste, können Sie die Kassette herausnehmen. Zuvor ist es jedoch ratsam, sie zurückzuspulen.

Um das Programm wieder in den Arbeitsspeicher zu laden, legen Sie die zurückgespulte Kassette mit dem gesicherten Programm "TEST" in die Datasette und geben

LOAD "TEST"

ein. Der Computer gibt daraufhin die Meldung

PRESS PLAY ON TAPE

aus. Jetzt drücken Sie die PLAY-Taste - auf keinen Fall jedoch die RECORD-Taste! - bis das Programm geladen ist und READY erscheint. An der Datasette befindet sich ein Zählwerk, das Sie unbedingt benutzen sollten, wenn Sie mehrere Programme hintereinander abspeichern. Es gibt hier nämlich keinen Schutz vor versehentlichem Überschreiben.

LOAD und SAVE funktionieren bei Kassettenbetrieb auch ohne Angabe eines Programmnamens; LOAD liest dann automatisch das nächstfolgende Programm ein. Befinden sich mehrere Programme auf einer Kassette, empfiehlt es sich jedoch, Namen zu verwenden. Bei LOAD "Name" läuft das Band dann solange, bis ein Programm mit dem entsprechenden Namen aufgefunden wird.

Bei Verwendung eines Diskettenlaufwerks ist der Vorgang ähnlich zu handhaben. Sie benötigen dazu entweder eine formatierte Diskette, auf der noch genügend Platz für Ihr Programm vorhanden ist, oder müssen eine Diskette neu formatieren. Achtung! Verwenden Sie dazu nur solche Disketten, die entweder fabrikneu sind oder solche, auf deren Inhalt Sie unbedingt verzichten können. In der Regel ist jede Aufzeichnung, die sich bereits auf einer Diskette befindet, nach dem Neuformatieren hoffnungslos verloren.

Im Gegensatz zum Band erhält jede Diskette einen Namen und eine Identifikationsbezeichnung (ID), die beide beim Formatieren angegeben werden müssen. Als Beispiel nehmen wir an, der Name sei "PROGRAMME" und die ID sei 01. Um eine Diskette zu formatieren, legen Sie sie ins Laufwerk und geben

HEADER "PROGRAMME",D0,I01

ein. Es erscheint daraufhin die Meldung

ARE YOU SURE? (Sind Sie sicher?)

Geben Sie nun Y (ja) oder N (nein) ein. Dies ist eine ins Programm eingebaute letzte Sicherheit, um Sie vor einem unbeabsichtigten Löschen der Diskette zu bewahren. Antworten Sie mit Y, dreht sich das Laufwerk etwa eine Minute lang, während die Diskette formatiert wird. Danach ist sie für den Einsatz auf dem PC 128 vorbereitet und sie können damit die folgenden Beispiele ausprobieren.

Auch das Abspeichern von Programmen ist ganz einfach; wir benötigen dazu nur die Anweisung DSAVE. Angenommen, wir verwenden wieder den Namen "TEST", vorausgesetzt, es findet sich kein anderes Programm mit gleichem Namen auf der Diskette, dann geben wir ein:

DSAVE "TEST"

Daraufhin wird das Programm auf Diskette geschrieben. Das Laden ist ebenso einfach; wir benötigen dazu die Anweisung DLOAD. Wenn Sie eingeben

DLOAD "TEST"

befindet sich das Programm wieder im Arbeitsspeicher und Sie können es normal mit RUN ausführen. Es besteht aber auch die Möglichkeit, es mit einem Befehl zu laden und gleich zu starten. Einen solchen Autostart gibt man folgendermaßen ein

RUN "TEST"

Jede Diskette enthält außerdem ein Inhaltsverzeichnis, in dem alle Programme und sonstige Dateien aufgeführt sind. Sie können es mit

DIRECTORY

auf dem Bildschirm ausgeben. Ein BASIC-Programm, das sich vielleicht gerade im Arbeitsspeicher befindlichet, wird dabei nicht zerstört. Überflüssige Dateien können Sie mit dem Befehl

SCRATCH "Name"

löschen, wobei "Name" für den betreffenden Dateinamen steht. Auch hier erfolgt zunächst wieder die Sicherheitsabfrage

ARE YOU SURE?

bevor der eigentliche Löschvorgang beginnt. Manchmal ist es vorteilhaft, einen Dateinamen umzubenennen. Dies geschieht mit dem Befehl

RENAME "Alter Name" TO "Neuer Name"

Soll z.B. unsere Datei "TEST" in "PROBE" umbenannt werden, geben Sie ein

RENAME "TEST" TO "PROBE"

Sie haben jetzt die wichtigsten Disketten- und Kassettenoperationen kennengelernt, die Sie zum Abspeichern und Laden von BASIC-Programmen benötigen. Wir werden uns jedoch im weiteren Verlauf dieses Buches noch mit anderen Befehlen zur Verwaltung von sequentiellen und relativen Dateien befassen.

2 Grundzüge der BASIC-Programmierung

Im letzten Kapitel haben wir uns mit den einfachsten Grundbegriffen für den Aufbau von BASIC-Programmen beschäftigt. Sie wissen jetzt, was die Programmiersprache BASIC ist, können im Direkt- und im Programmodus einfache Meldungen ausgeben und haben einige nützliche Hilfsmittel zur Erstellung, Korrektur und Sicherung von Programmen kennengelernt. In diesem Kapitel erfahren Sie nähere Einzelheiten über Datentypen, Variablen und die wichtigsten Ein- und Ausgabeoperationen und lernen, damit umzugehen.

2.1 Datentypen

Der Computer kann eine Vielzahl von Daten verarbeiten, vorausgesetzt, er kennt alle dazu erforderlichen Informationen. Diese können entweder im BASIC-Programm selbst oder auf einem Datenträger (z.B. Kassette oder Diskette) vorgegeben sein.

Der Begriff "Daten" ist sehr umfassend. Wenn Sie beispielsweise das Telefonbuch aufschlagen, finden Sie darin die Daten sämtlicher Fernsprechteilnehmer, d.h. Nach- und Vorname, Straße und Hausnummer und natürlich die Rufnummer. Darüberhinaus sind die Teilnehmer in alphabetischer Reihenfolge geordnet, damit Sie sie leicht auffinden können.

Ein ähnliches Verzeichnis, wie es das Telefonbuch darstellt, können Sie durchaus auch in Ihrem Computer speichern und verarbeiten, selbst wenn es nur die Adressen und Telefonnummern Ihrer Freunde und Bekannten enthält. Eine solche Ansammlung verschiedener Daten, gleich welcher Art, bezeichnet man als Datei.

Wenn Sie Ihren Computer im Privatbereich nutzen wollen, können Sie sich z.B. ein Programm schreiben, das Ihr Haushaltsgeld verwaltet und die Einnahmen, Ausgaben usw. in einer Datei ablegt; oder ein Programm, das Ihre Schallplattensammlung registriert oder die Autokosten auflistet. Im professionellen Bereich laufen heutzutage Buchhaltung und Lagerhaltung meist über die elektronische Datenverarbeitung mit teilweise riesigen Dateien, die auch als Datenbanken bezeichnet werden.

Auf die eigentliche Dateiverwaltung werden wir später noch zu sprechen kommen. Im Augenblick geht es nur darum, welche Art von Daten oder Datentypen Ihr Computer verarbeiten kann.

2.1.1 Strings

Bereits im letzten Kapitel haben wir mit Zeichenketten gearbeitet, die man in der Fachsprache als Strings bezeichnet. Die Anweisung

```
PRINT "GUTEN TAG, ICH BIN DER PC 128"
```

gibt beispielsweise den String

```
GUTEN TAG, ICH BIN DER PC 128
```

auf dem Bildschirm aus, was durch das Befehls- oder BASIC-Schlüsselwort PRINT ausgelöst wird. Hier einige weitere Beispiele für Strings:

```
"HANS SCHMIDT"
"AUGUSTENSTRASSE 48"
"VIELE GRUESSE VOM COMPUTER"
"PFEFFER"
"TOMATEN"
"123.456"
```

Strings sind in der Regel eine Aneinanderreihung von Buchstaben, Satzzeichen und Ziffern, wie Sie sie auf einer Schreibmaschine tippen. Damit der Computer erkennt, wo der String beginnt und wo er endet, ist er in Anführungszeichen eingeschlossen. Aus diesem Grund dürfen in einem String selbst keine Anführungszeichen auftreten. Soll ein Wort besonders hervorgehoben werden, verwendet man statt dessen meist den Apostroph.

Hier ein Beispiel:

Falsch:

    "SIE SAGTE: "MEIN NAME IST HASE, ICH WEISS VON NICHTS"."

Richtig:

    "SIE SAGTE: 'MEIN NAME IST HASE, ICH WEISS VON NICHTS.'"

Als wir im letzten Kapitel lernten, Programme abzuspeichern und zu laden, haben wir bereits mit Strings gearbeitet. So ist beispielsweise in der Anweisung

    DLOAD "TEST"

der Programmname "TEST" ein String. Er wird von dem Schlüsselwort DLOAD benötigt, um das richtige Programm unter dem Namen TEST von Diskette zu laden.

Aus programmiertechnischen Gründen verwendet man manchmal auch Strings, die keine Zeichen oder nur Leerzeichen (Blanks) enthalten. Z.B. ist

    ""

ein Nullstring, weil er keine Zeichen enthält und

    "   "

ist ein Leerstring mit 14 Blanks, der nur an den Anführungszeichen zu erkennen ist.

Jedes Zeichen, das in einem String dargestellt werden kann, besitzt einen entsprechenden Zahlencode, der im Computer gespeichert ist. Dieser Code ist genormt und heißt ASCII-Code. So entspricht z.B. der Buchstabe A dem ASCII-Code 65, der Buchstabe B dem ASCII-Code 66 usw. (Siehe dazu auch ASCII-Code-Tabelle im Anhang!)

Entdeckt nun der Computer hinter dem Schlüsselwort PRINT ein Anführungszeichen, so weiß er, daß er einen String auf dem Bildschirm ausgeben soll. Ist der Wert in der nächsten Speicherzelle (Byte) eine 65, gibt er folglich den Buchstaben A aus.

Fast alle Computer arbeiten heute mit dem ASCII-Code. Dies hat den Vorteil, daß sie Daten untereinander austauschen können.

2.1.2 Fließkommazahlen

Fließkommazahlen sind neben den Strings ein Datentyp, mit dem der Computer echt "rechnen", d.h. mathematische Operationen durchführen kann. Fließkommazahlen werden deshalb vom Computer in einer anderen Form intern verwaltet als Strings.

Vielleicht fragen Sie sich, warum unter den Beispielen für Strings auch "123.456" aufgeführt ist. Zwar ist der Inhalt dieses Strings offensichtlich ein Zahlenwert, der Computer betrachtet ihn aber nicht als Zahl, da er in Anführungszeichen eingeschlossen ist. Dieser String enthält lediglich die äquivalenten ASCII-Zeichen für 1, 2, 3, 4, 5 und 6. In diesem Format ist der Computer nicht in der Lage, arithmetische Berechnungen durchzuführen.

Geben wir also 123.456 ohne Anführungszeichen ein, so wird dieser Wert als Zahl betrachtet und intern in einem speziellen Binärformat verwaltet. Zahlen im Sinne des Computers dürfen nur aus den Ziffern 0 bis 9 und eventuell einem Vorzeichen und einem Dezimalpunkt (kein Komma!) bestehen. Sehr große und sehr kleine Zahlen werden in Exponentialschreibweise angegeben, wobei der Wert durch ein großes E mit einem zweistelligen Zehnerexponenten abgeschlossen wird. Hier einige Beispiele für zulässige Zahlenwerte:

```
1.5
652
-1546.98006
.044736524
1.54E+21
5.987543E-07
```

Unzulässig dagegen sind z.B.:

7,52	richtig:	7.52
-65,765,879.8	richtig:	-65765879.8
34.652,87	richtig:	34652.87
3.43-	richtig:	-3.43

Ebenso wie Strings, können mit PRINT auch Zahlen auf dem Bildschirm ausgegeben werden:

```
PRINT 45.789
 45.789

READY.
```

oder

```
PRINT -27
 -27

READY.
```

Der PC 128 stellt Fließkommazahlen in einem möglichst optimalen Format dar. Deshalb werden Werte größer gleich 9000000000 und kleiner als .01 in Exponentialschreibweise ausgegeben, wie z.B.

```
PRINT 1356789776655443
 1.35678978E+15

READY.
```

oder

```
PRINT 0.0000552
 5.52E-05

READY.
```

Der PC 128 kann nur Zahlenwerte mit maximal 9 signifikanten Ziffern verarbeiten. Größere Zahlen werden auf insgesamt neun Ziffern gerundet. Beachten Sie auch, daß der zulässige Wertebereich nur zwischen ca. 1E-39 und 1E+38 liegt. Kleinere Werte werden als 0 wiedergegeben, bei größeren erfolgt eine Überlauf-Fehlermeldung, wie im folgenden Beispiel:

```
PRINT 1E+90

?OVERFLOW ERROR
READY.
```

2.1.3 Integerzahlen

Neben den Fließkommazahlen gibt es noch die Integerzahlen. Sie dürfen nur ganzzahlige Werte im Bereich zwischen -32768 und +32767 enthalten.

Integerzahlen benötigen weniger Speicherplatz als Fließkommazahlen. Bei Commodore-Rechnern haben sie, abgesehen von Feldern (Arrays) mit geringerem Speicherbedarf, nur eine untergeordnete Bedeutung. Ansonsten belegen sie genausoviel Speicherplatz wie eine Fließkommazahl, von dem allerdings ein Teil unbenutzt bleibt. Auch die Rechenzeit mit Integerzahlen ist, im Gegensatz zu anderen Rechnern, bei Commodore etwa gleich groß wie die mit Fließkommazahlen, was im Betriebssystem begründet ist. Deshalb wollen wir uns an dieser Stelle nicht näher mit Integerzahlen befassen.

2.2 Rechnen mit BASIC

In BASIC ist das Rechnen mit Zahlen sehr einfach. Die Eingabe erfolgt in ähnlicher Form wie in der Algebra, wie wir es aus der Schule kennen.

Mit PRINT können wir auch arithmetische Ausdrücke, wie z.B. 4 + 9 oder 65 - 3 auswerten. Das Zeichen, das die betreffende Operation angibt, heißt Operator. In diesem Fall dient der Operator + zur Addition und der Operator / zur Division. Die Werte, mit denen die Operation stattfinden soll, heißen Operanden. Das Ergebnis, das man durch Ausführung der Operation enthält, bezeichnet man als Wert der Operation.

Entdeckt nun der Computer hinter einer PRINT-Anweisung eine arithmetische Operation, so führt er sie aus und gibt deren Wert, also das Ergebnis, auf dem Bildschirm aus. Hier einige Beispiele:

```
PRINT 3 + 4
 7

READY.

PRINT 100 - 2
 98

READY.

PRINT -10/3
 -3.33333333

READY.

PRINT 45 * 5
 225

READY.
```

Der Vollständigkeit halber wollen wir uns hier auch kurz mit den wissenschaftlichen Funktionen befassen. Beginnen wir mit der Exponentiation:

```
PRINT 3 ^ 2
 9

READY.
```

In diesem einfachen Fall wären wir auch mit einer Multiplikation von 3 * 3 zum gleichen Ergebnis gekommen. Wir sind aber nicht nur auf geradzahlige Exponenten beschränkt, wie das folgende Beispiel zeigt:

```
PRINT 45 ^ 1.63
 495.14956

READY.
```

Beachten Sie, daß das Potenzierzeichen "^" dem senkrecht nach oben gerichteten Pfeil auf der Tastatur entspricht.

Umgekehrt können wir mit SQR auch die Quadratwurzel ziehen:

```
PRINT SQR(2)
 1.41421356

READY.
```

Folgende Exponentiation führt zum gleichen Ergebnis, da die Quadratwurzel einem Exponenten von 0.5 entspricht:

```
PRINT 2 ^ .5
 1.41421356

READY.
```

Hier noch ein kurzer Überblick über die trigonometrischen und logarithmischen Funktionen. X steht dabei jeweils für das Argument.

SIN(X)	Sinus
COS(X)	Cosinus
TAN(X)	Tangens
ATN(X)	Arcustangens
EXP(X)	Potenz zur Zahl e
LOG(X)	natürlicher Logarithmus

Zu bemerken ist noch, daß die Winkel für trigonometrische Berechnungen immer im Bogenmaß angegeben werden müssen! Beispiel: Es ist der Sinuswert für einen Winkel von 45 Grad zu berechnen. Da die Sinusfunktion nur Winkel im Bogenmaß verarbeitet, muß der Winkel zunächst vom Gradmaß ins Bogenmaß umgerechnet werden:

45 Grad = 45 * Pi / 180 (Bogenmaß) = .785398163

Dann folgt

```
PRINT SIN(.785398163)
 .707106781

READY.
```

Pi steht hier für die Zahl 3.14159265, die auf Ihrem Rechner auch per Tastendruck abrufbar ist.

In BASIC können Sie auch arithmetische Ausdrücke berechnen, die mehrere Operationen erfordern:

```
PRINT 3 + 4 + 19 - 10
 16

READY.
```

und

```
PRINT 7 + 2*9 - 6/3
 23

READY.
```

Die Berechnungen werden von links nach rechts durchgeführt, wobei folgende Prioritäten zu beachten sind, die wir auch in der Algebra vorfinden:

+ und -	Addition und Subtraktion (niedrigste Priorität)
* und /	Multiplikation und Division
^	Potenzierung (höchste Priorität)

Für unser letztes Beispiel bedeutet das, daß zur Zahl 7 das Produkt von 2*9 = 18 addiert wird. Von diesem Ergebnis wird dann der Quotient 6/3 = 2 subtrahiert; das führt zum Endergebnis von 23.

BASIC kann auch Klammerrechnungen ausführen, mit denen die Prioritäten umgangen werden können. Dabei wird jeweils der Klammerinhalt vorrangig ausgewertet.

Wir wollen uns nochmals das letzte Beispiel ansehen, jetzt aber etwas umgeformt:

```
PRINT (7+2) * (9-6) / 3
 9

READY.
```

Hier werden zunächst die beiden Klammerinhalte 9 bzw. 3 miteinander multipliziert und dann das Produkt durch 3 dividiert.

Aber auch unsere Sinusberechnung (s.o.) können wir vereinfachen, ohne erst das Bogenmaß von 45 Grad separat berechnen zu müssen. Für das Argumant X kann nämlich auch ein arithmetischer Ausdruck stehen:

```
PRINT SIN (45 * 3.14159265 / 180)
 .707106781

READY.
```

Vor der Berechnung des eigentlichen Sinuswertes wird also zunächst der Klammerinhalt als dessen Argument ausgewertet, d.h,Gradmaß in Winkelmaß umgerechnet.

2.3 Weitere Möglichkeiten mit PRINT

Bisher haben wir mit PRINT einen String oder eine Zahl auf dem Bildschirm ausgegeben und schließlich die Auswertung eines arithmetischen Ausdrucks mit eingeschlossen. Darüberhinaus kann man aber mit PRINT noch einige nützliche Ausgabetechniken realisieren.

Bei den besprochenen Beispielen gab jede PRINT-Anweisung immer nur einen Wert aus. Es besteht jedoch auch die Möglichkeit, den Inhalt mehrerer Strings in einer Bildschirmzeile darzustellen. Um dies zu demonstrieren, schreiben wir am besten ein kleines BASIC-Programm:

```
10 PRINT "ZU"
20 PRINT "SAM"
30 PRINT "MEN"
```

Nachdem Sie dieses Programm mit RUN gestartet haben, erscheinen die drei Strings in der uns bereits bekannten Form:

```
ZU
SAM
MEN

READY.
```

Jetzt ändern wir das Programm geringfügig ab, indem wir Zeile 10 und 20 mit einem Semikolon abschließen:

```
10 PRINT "ZU";
20 PRINT "SAM";
30 PRINT "MEN"
```

Nach Ausführung dieses Programms erhalten wir als Bidschirmausgabe:

```
ZUSAMMEN
READY.
```

Wir sehen also, daß verschiedene Strings aus verschiedenen PRINT-Anweisungen aneinandergefügt werden, wenn sie jeweils mit einem Semikolon abschließen. Das gleiche Resultat können wir noch einfacher erhalten:

```
PRINT "ZU";"SAM";"MEN"
ZUSAMMEN

READY.
```

Hier stehen die einzelnen Strings in einer PRINT-Anweisung, wobei sie jeweils durch Semikolons getrennt sind.

Ihr PC 128 bietet außerdem die Möglichkeit der festen Tabellierung, wobei die Werte im Abstand von jeweils 10 Zeichen beginnen.

Zu diesem Zweck müssen sie durch Kommas getrennt sein:

```
PRINT "AUS","EIN","AN","DER"
AUS       EIN       AN        DER

READY.
```

Ein Tabellator mit variablen Abständen steht mit der TAB-Anweisung zur Verfügung:

```
PRINT TAB(3)"ABC";TAB(20)"DEF"
   ABC              DEF

READY.
```

Der Klammerwert innerhalb der TAB-Anweisung gibt die jeweilge Position in der Zeile an, beginnend mit Position 0 am linken Rand. Im diesem Beispiel beginnt der erste String bei Position 3 und der zweite bei Position 20. Sämtliche PRINT-Anweisungen gelten auch für Zahlen vom Datentyp Fließkomma oder Integer bzw. für Werte von berechneten Ausdrücken, wie wir sie bereits kennengelernt haben. Bei positiven Werten wird statt des Vorzeichens immer ein Leerzeichen freigelassen:

```
PRINT 5,-4,3
5        -4        3

READY.
```

Steht zwischen den Zahlen allerdings ein Semikolon, so bleibt ein zusätzliches Leerzeichen frei, um die Übersichtlichkeit zu erhöhen:

```
PRINT 5;-4;3
5 -4  3

READY.
```

Abschließend noch ein Programm, das für verschiedene Schüler den Namen zusammen mit der Punktzahl und der Note einer Klassenarbeit ausgibt:

```
10 PRINT "NAME","PUNKTE","NOTE"
20 PRINT
30 PRINT "HANS",45,3
40 PRINT "NICOLE",40,4
50 PRINT "STEFAN",50,2.7
60 PRINT "CHRISTIAN",38,4.3
70 PRINT "URSULA",60,2
```

Nach Eingabe von RUN erscheint folgende Liste auf dem Bildschirm:

```
NAME       PUNKTE    NOTE

HANS         45        3
NICOLE       40        4
STEFAN       50        2.7
CHRISTIAN    38        4.3
URSULA       60        2

READY.
```

2.4 Variablen

Wir wissen bereits, daß der Computer einen RAM-Speicher besitzt, d.h. einen flüchtigen Speicher, dessen Inhalt sich ständig ändern kann und beim Ausschalten des Computers verloren geht. Ein Großteil des RAM-Speichers dient zur Ablage von BASIC-Programmen und allen dazugehörigen Informationen.

Weiter haben wir einige einfache Programme kennengelernt, die in numerierte BASIC-Zeilen aufgeteilt sind. Unsere Beispiele im Direkt- und im Programmodus behandelten bisher ausschließlich die Ausgabe von Strings auf dem Bildschirm und die Berechnung arithmetischer Ausdrücke. Die Werte, die zu berechnen bzw. auszugeben waren, standen unmittelbar hinter der PRINT-Anweisung.

Auch kennen wir schon die verschiedenen Datentypen, nämlich Strings für Zeichenketten, Fließkommazahlen und Integerzahlen.

Intern verwaltet der Computer jeden dieser Datentypen auf unterschiedliche Weise. So werden Strings in ASCII-Zeichen, Fließkommazahlen in einem Binärformat und Integerzahlen in einer kurzen Binärschreibweise abgelegt. Zu diesem Zweck muß der Computer für jeden einzelnen Wert einen Speicherbereich reservieren.

Betrachten wir nun nochmals unser allererstes Beispiel, das wir dieses Mal allerdings im Programmodus schreiben wollen:

```
10 PRINT "GUTEN TAG, ICH BIN DER PC 128"
```

Wenn Sie diese Programmzeile eintippen und die RETURN-Taste drücken, ordnet der Computer ihr einen bestimmten Speicherplatz zu, in dem auch die auszugebende Meldung enthalten ist. Sofern wir den Text nicht ändern, erscheint jedesmal bei Ausführung von Zeile 10 die gleiche Meldung. Wenn wir den Text nun anstelle von einem Mal zehnmal hintereinander ausgeben wollen, brauchen wir ein Programm, das zehn Zeilen mit jeweils der gleichen PRINT-Anweisung enthält. Dabei könnten die Zeilen z.B. von 10 bis 100 in Zehnerschritten durchnumeriert werden, eine wahrhaftig umständliche Angelegenheit!

BASIC hat jedoch eine Möglichkeit, bei der wir den String nur einmal definieren müsssen und ihn mehrfach hintereinander mit einer wesentlich kürzeren Anweisung ausgeben können. Dazu ordnen wir den String einer Variablen zu, die dann nach Belieben aufgerufen werden kann:

```
10 LET A$ = "GUTEN TAG, ICH BIN DER PC 128"
20 PRINT A$
30 PRINT A$
40 PRINT A$
```

Wenn Sie dieses Programm ausführen, erscheint dreimal die gleiche Meldung auf dem Bildschirm. In Zeile 10 haben wir den Text in einer Stringvariablen mit dem Namen A$ abgelegt, was mit der Zuordnungsanweisung LET geschieht. Der Computer führt ein internes Verzeichnis, in dem genau angegeben ist, an welcher Stelle sich der Inhalt von A$ im Speicher befindet und wieviele Zeichen der String umfaßt. Jedesmal, wenn nun bei der Abarbeitung des Programms A$ auftaucht, sucht der Computer diese Variable im Verzeichnis und bearbeitet ihren Inhalt gemäß der entsprechenden Anweisung. In unserem Beispiel wird sie insgesamt dreimal durch PRINT aufgerufen und auf dem Bildschirm ausgegeben.

Ähnlich wie bei der Zuweisung eines Strings zu einer Variablen kann auch bei numerischen Ausdrücken verfahren werden, wie das folgende Beispiel zeigt:

```
10 LET A = 3
20 LET B = 5
30 LET C = A * B
40 PRINT C
```

Nach Ausführung dieses Programms erscheint als Ergebnis die Zahl 15 auf dem Bildschirm. Wie es dazu kommt, wollen wir nun einmal näher untersuchen.

In Zeile 10 erhält die Fließkommavariable A den Wert 3 und in Zeile 20 die Fließkommavariable B den Wert 5. Beide Werte werden im internen Zahlenformat im Speicher abgelegt und können mit Hilfe der Variablennamen A bzw. B wieder aufgefunden werden. Dies geschieht in Zeile 30. Hier entsteht eine neue Variable C, nachdem die Inhalte der Variablen A und B miteinander multipliziert wurden. Diese Variable C enthält nun das Produkt aus A und B und hat ebenfalls einen festen Speicherbereich. Schließlich wird ihr Inhalt in Zeile 40 aufgerufen und auf dem Bildschirm ausgegeben.

In diesem Beispiel haben wir die Zuordnung eines Wertes zu einer Variablen mit LET durchgeführt. Diese Anweisung stammt noch aus der Anfangszeit von BASIC und ist bei modernen BASIC-Dialekten nicht mehr erforderlich. Auch wir wollen die LET-Anweisung in diesem Buch nicht weiter verwenden und sie künftig einfach weglassen. Dann sieht unser Beispielprogramm folgendermaßen aus:

```
10 A = 5
20 B = 3
30 C = A * B
40 PRINT C
```

Hier nun noch etwas zu den Variablennamen: Wir kennen bereits die drei Datentypen String, Fließkomma und Integer, von denen jeder einen bestimmten Variablennamentyp besitzt. Wir wissen, daß Variablennamen für Strings immer mit einem $-Zeichen enden, das bei den Fließkommavariablen fehlt. Dies ist auch genau das Unterscheidungsmerkmal zwischen den beiden Typen. Namen für Integervariablen enden mit einem %-Zeichen. Hier nochmals eine Zusammenfassung:

Variablentyp	**Merkmal**
String	$
Fließkomma	keines
Integer	%

Jeder Variablenname des gleichen Typs darf in einem Programm nur einmal vorkommen. Eine erneute Zuordnung mit einem anderen Wert ist jedoch möglich, wobei der alte Wert verlorengeht. Hier ein Beispiel:

```
10 A = 2
20 A = 100
30 A = 3 * 5
40 PRINT A
```

In Zeile 10 erhält die Variable A den Wert 2 und in Zeile 20 den Wert 100, wobei der Wert 2 aus der Zuordnung in Zeile 10 verlorengeht. Das gleiche geschieht nochmals in Zeile 30, wo die Variable A das Produkt aus 3 und 5, also 15 erhält.

Anders ist es, wenn der gleiche Variablenname in den verschiedenen Typen erscheint, wie z.B.:

```
10 A = 6.78
20 A% = 2
30 A$ = "HALLO"
40 PRINT A
50 PRINT A%
60 PRINT A$
```

Nach Ausführung des Programms erscheint:

```
 6.78
 2
HALLO
READY.
```

Wir sehen also, daß der gleiche Variablenname unter den verschiedenen Typenbezeichnungen existieren kann.

Bei der Wahl der Variablennamen ist außer der Typenbezeichnung noch folgendes zu beachten: Im Commodore-BASIC dürfen Variablennamen außer der Typenbezeichnung maximal zwei Zeichen umfassen, wobei der erste auf jeden Fall ein Buchstabe zwischen A und Z sein muß. Das zweite Zeichen ist nicht zwingend erforderlich und kann entweder ebenfalls aus einem Buchstaben oder aus den Ziffern 0 bis 9 bestehen. Hier ein paar Beispiele zulässiger Variablennamen:

AB	Z1$	CC$	U7%
TY$	B%	R0$	UA

Zwar verarbeitet der Computer auch Variablennamen mit mehr als zwei Zeichen, dabei sind dann aber nur die ersten beiden Zeichen signifikant. Außerdem ist bei der Wahl eines längeren Variablennamens zu beachten, daß es keine Verwechslungen mit anderen Namen gibt, wie im folgenden Beispiel:

```
10 LAGER$ = "BESTAND"
20 LATTE$ = "BRETT"
```

Beide Variablennamen beginnen mit LA. Da der Computer nur die beiden ersten Zeichen auswertet und die anderen Zeichen für ihn ohne Bedeutung sind, haben beide Ausdrücke den Variablennamen LA$. In diesem Programm würde der String BESTAND in Zeile 20 durch BRETT überschrieben werden.

Sie sollten den Variablen möglichst sinnvolle Namen zuteilen, was gleichzeitig der Übersichtlichkeit des Programms dient. Die Variationsmöglichkeiten sind allerdings, wie das obige Beispiel zeigt, im Commodore-BASIC auf nur zwei Zeichen beschränkt, will man keine Verwechslungen riskieren. Dies ist fraglos ein Nachteil, den viele andere BASIC-Versionen nicht haben, da sie teilweise 40 Zeichen lange Variablennamen zulassen.

Einige Variablennamen auf Ihrem PC 128 sind bereits für bestimmte Zwecke reserviert und dürfen von Ihnen nicht für andere Zwecke verwendet werden:

TI und TI$	zur Abfrage der internen Uhr
ST	Statusvariable für I/O-Operationen
ER, EL und ERR$	zur Fehlerbehandlung
DS, DS$	zur Abfrage von Fehlern bei Diskettenoperationen
FN	zum Aufruf selbstdefinierter Funktionen

Als weiteres sind auch Variablennamen unzulässig, die BASIC-Schlüsselwörter enthalten, wie z.B:

ABLOAD   UPRINT$

Achten Sie darauf, daß Daten- und Variablentyp übereinstimmen, d.h. daß Sie einer Stringvariablen keinen numerischen Wert zuweisen und umgekehrt. Wäre dies der Fall, wie im folgenden Beispiel

```
A$ = 5.25
```

entsteht die Fehlermeldung

```
?TYPE MISMATCH ERROR
```

Die richtige Eingabe wäre gewesen:

```
A = 5.25
```

2.5 Die INPUT-Anweisung

Wir wissen bereits, daß wir Variablen bestimmte Werte zuordnen können. Bisher geschah dies durch Zuordnungsanweisungen mit oder ohne LET.

Oft ist es jeoch erforderlich, während des Programmablaufs Werte anzufordern, die über die Tastatur eingegeben und dann einer Variablen zugeordnet werden. Dies geschieht mit der INPUT-Antweisung. Probieren Sie einmal das folgende kleine Programm aus:

```
10 INPUT A
20 PRINT A
```

Wenn Sie dieses Programm mit RUN starten, erscheint auf dem Bildschirm ein Fragezeichen und dahinter der blinkende Cursor.

Geben Sie jetzt eine beliebige Fließkommazahl ein und drücken Sie die RETURN-Taste wie im folgenden Beispiel:

```
? 145    <RETURN>
  145
READY.
```

Hierbei fordert die INPUT-Anweisung in Zeile 10 eine Fließkommazahl an, die, nachdem sie eingegeben und durch RETURN abgeschlossen wurde, der Fließkommavariablen A zugeordnet wird. Zur Kontrolle steht die PRINT-Anweisung in Zeile 20, die die betreffende Zahl wieder auf dem Bildschirm ausgeben muß.

INPUT ist nicht nur auf Fließkommawerte beschränkt, sondern funktioniert mit jedem Datentyp gleichermaßen. Hier einige weitere Beispiele:

```
INPUT B%
INPUT XY$
INPUT A, B$, C, DY
```

Betrachten wir einmal das letzte Beispiel genau, bei dem insgesamt vier Werte für verschiedene Variablen angefordert werden:

```
10 INPUT A, B$, C, DY
20 PRINT A
30 PRINT B$
40 PRINT C
50 PRINT DY

RUN
? 2,PRIMA,25,3.14
 2
PRIMA
 25
 3.14

READY.
```

Hier werden nacheinander Werte für die Fließkommavariable A, die Stringvariable B$ sowie die Fließkommavariablen C und DY angefordert. Bei der Eingabe müssen sie durch Kommas getrennt werden, damit der BASIC-Interpreter erkennt, wo sie beginnen und enden.

Strings müssen in diesem Fall nicht unbedingt in Anführungszeichen eingegeben werden. Wenn Sie allerdings statt einem reinen Zahlenwert unzulässige Zeichen, z.B. Buchstaben, eintippen, gibt der Computer

```
?REDO FROM START
?
```

aus. Der erfaßte Wert wird ignoriert und das untere Fragezeichen fordert Sie erneut zur Eingabe auf. Diese Meldung erscheint auch, wenn Sie versucht haben, eine Integerzahl außerhalb ihres zulässigen Bereiches (-32768 bis 32767) einzugeben.

Wenn nun während des Programmablaufs plötzlich ein Fragezeichen und der Cursor auftauchen, weiß der Benutzer zwar, daß er einen Wert eingeben soll, er weiß jedoch häufig nicht, um welchen Wert es sich dabei handelt. BASIC sieht deshalb die Möglichkeit vor, daß vor der Anforderung zunächst ein Hinweis auf dem Bildschirm steht:

```
10 INPUT "BITTE BELIEBIGE ZAHL EINGEBEN"; Z
20 PRINT "SIE HABEN SOEBEN DIE ZAHL"; Z; " EINGEGEBEN"

RUN
BITTE BELIEBIGE ZAHL EINGEBEN? 100
SIE HABEN SOEBEN DIE ZAHL 100 EINGEGEBEN

READY.
```

Beachten Sie die PRINT-Anweisung in Zeile 20, in der das Programm insgesamt drei Werte ausgibt. Zunächst handelt es sich dabei um den String "SIE HABEN SOEBEN DIE ZAHL", dann um die eingegebene Zahl Z und schließlich um den zweiten String "EINGEGEBEN".

Zum Schluß wollen wir noch ein kleines Programm betrachten, das das Volumen eines Quaders berechnet. Dies geschieht nach der Formel

Volumen = Länge * Breite * Höhe

Wie wir bereits wissen, erkennt das BASIC 7.0 nur Variablennamen mit höchstens zwei Zeichen. Wir vereinbaren deshalb folgende Variablennamen:

V für Volumen
L für Länge
B für Breite
H für Höhe

Hier nun das komplette Programm:

```
10 REM VOLUMENBRECHNUNG
20 INPUT "LAENGE"; L
30 INPUT "BREITE"; B
40 INPUT "HOEHE"; H
50 V = L * B * H
60 PRINT "DAS VOLUMEN BETRAEGT"; V
```

Dieses Programm enthält die REM-Anweisung. REM steht für engl. "Remark" oder Kommentar. Entdeckt der Computer eine REM-Anweisung, so übergeht er sie und ignoriert alle nachfolgenden Zeichen. REM-Zeilen dienen in erster Linie der Übersichtlichkeit von Programmen und können jeden beliebigen Text enthalten. In unserem Fall zeigt die REM-Anweisung in Zeile 10 an, daß wir es mit einem Programm zur Volumenberechnung zu

tun haben. Ein Programm kann aber beliebig viele REM-Anweisungen an beliebigen Stellen enthalten.

In den Zeilen 20 bis 40 werden nacheinander Länge, Breite und Höhe eingegeben und den Variablen L, B und H zugeordnet. Zeile 50 berechnet dann das Volumen V, indem es diese drei Werte miteinander multipliziert. Zeile 60 gibt schließlich das Ergebnis mit einem entsprechenden Hinweis aus. Hier ein Probelauf:

```
LAENGE? 20
BREITE? 10
HOEHE? 6
DAS VOLUMEN BETRAEGT 1200

READY.
```

3 Programmschleifen und Verzweigungen

In den vorangehenden Kapiteln haben wir Programme kennengelernt, die ausschließlich von vorn bis hinten, d.h. von der ersten bis zur letzten Zeile, abgearbeitet wurden und dann endeten. Ein Computer hätte aber nur einen kleinen Einsatzbereich, wenn er lediglich Programme dieser Art verarbeiten könnte.

Außer zu Übungszwecken gibt es aber kaum ein Programm, das sich mit den bisher vorgestellten Anweisungen begnügt. Vielmehr werden viele Anweisungen wiederholt abgearbeitet oder nur dann ausgeführt, wenn eine bestimmte Bedingung erfüllt oder nicht erfüllt ist. Mit derartigen Anweisungen wollen wir uns in diesem Kapitel beschäftigen.

Bei wiederholt ausgeführten Anweisungen spricht man auch von Programmschleifen, die nicht nur aus einer, sondern aus einer Vielzahl von Einzelanweisungen bestehen können. Häufig enthalten diese Schleifen eine Abbruchbedingung, d.h. sie werden solange wiederholt ausgeführt, bis diese Bedingung erfüllt ist. Durch diese Programmstruktur ist der Computer in der Lage, regelrechte Entscheidungen zu treffen.

Im folgenden wollen wir uns nun mit den verschiedenen Wiederholungs- und Verzweigungsanweisungen befassen, die das BASIC 7.0 zur Verfügung stellt.

3.1 FOR...NEXT

Die wohl einfachste und bekannteste Wiederholungsanweisung ist die FOR...NEXT-Anweisung. Wir wollen sie anhand eines kleinen Beispieles einmal näher betrachten.

Angenommen, Sie wollen die Zahlen 1 bis 10 nacheinander auf dem Bildschirm ausgeben. Dazu gibt es zwei Möglichkeiten, von denen die erste so aussieht:

```
10 PRINT 1;
20 PRINT 2;
30 PRINT 3;
40 PRINT 4;
50 PRINT 5;
60 PRINT 6;
70 PRINT 7;
80 PRINT 8;
90 PRINT 9;
100 PRINT 10;
```

Jetzt starten wir dieses Programm mit RUN und erhalten

```
1 2 3 4 5 6 7 8 9 10

READY.
```

In diesem Fall mußten wir zehnmal die PRINT-Anweisung ausführen, um alle Zahlen zwischen 1 und 10 auszugeben. Nun wollen wir das Programm so umschreiben, daß wir hinter PRINT nicht eine Zahl, sondern eine Variable I ausgeben. Damit dieses Programm nicht allzuviel Platz einnimmt, wollen wir immer zwei durch einen Doppelpunkt getrennte Anweisungen in eine Programmzeile setzen. Theoretisch können es so viele Anweisungen sein, wie in eine Zeile passen. Eine Programmzeile darf im BASIC 7.0 maximal 80 Textzeichen umfassen, was beim 80-Zeichen-Modus eine Zeile und beim 40 Zeichen-Modus zwei Zeilen auf dem Bildschirm entspricht.

```
10 I = 1 : PRINT I;
20 I = 2 : PRINT I;
30 I = 3 : PRINT I;
40 I = 4 : PRINT I;
50 I = 5 : PRINT I;
60 I = 6 : PRINT I;
70 I = 7 : PRINT I;
80 I = 8 : PRINT I;
90 I = 9 : PRINT I;
100 I = 10 : PRINT I;
```

In diesem Beispiel haben wir jeweils die Werte 1 bis 10 der Variablen I zugeordnet, aber immer noch müssen wir für jeden Wert die Anweisungen schreiben.

Um dies zu erleichtern, verwenden wir jetzt die FOR...NEXT-Anweisung. Dann sieht unser Programm so aus:

```
10 FOR I = 1 TO 10
20 PRINT I;
30 NEXT I
```

I ist hier die Laufvariable, die von 1 bis 10 durchzählt. Alle Anweisungen, die zwischen FOR und NEXT stehen, werden so häufig ausgeführt, wie es in der FOR-Anweisung angegeben ist. Erinnern wir uns nochmals daran, daß hinter der PRINT-Anweisung ein Semikolon steht, damit die einzelnen Zahlenwerte nebeneinander ausgegeben werden. Fehlt das Semikolon, erscheinen die Zahlenwerte untereinander, jeweils in einer neuen Zeile.

Damit sind die Möglichkeiten der FOR...NEXT-Anweisung aber noch lange nicht erschöpft. Der Zusatz STEP ermöglicht uns, auch rückwärts oder in größeren Schritten zu zählen; fehlt er dagegen, werden Einerschritte angenommen. Betrachten wir folgendes Beispiel:

```
10 FOR I = 10 TO 1 STEP -1
20 PRINT I;
30 NEXT I
```

Nach RUN folgt

```
10 9 8 7 6 5 4 3 2 1

READY.
```

STEP -1 gibt an, daß in Einerschritten zwischen 10 und 1 rückwärts gezählt werden soll. STEP kann aber auch einen größeren positiven oder negativen Wert einnehmen, wie im folgenden Beispiel:

```
10 FOR I = 0 TO 50 STEP 10
20 PRINT I;
30 NEXT I
```

Nach RUN folgt

```
0 10 20 30 40 50

READY.
```

Hinter NEXT kann der Laufvariablenname I auch weggelassen werden, was u.a. eine geringfügige Erhöhung der Abarbeitungsgeschwindigkeit zur Folge hat.

Die Schrittweite muß nicht unbedingt ganzzahlig sein, wie das folgende Beispiel zeigt:

```
10 FOR I = .2 TO -.6 STEP -.2
20 PRINT I;
30 NEXT
```

Nach RUN folgt

```
.2 0 -.2 -.4 -.6

READY.
```

Die allgemeine Form der FOR...NEXT-Anweisung lautet:

FOR Variable = Anfangswert TO Endwert STEP Schrittweite
(Verschiedene Anweisungen)
NEXT Variable oder nur **NEXT**

In der FOR...NEXT-Anweisung können sowohl Anfangs- und Endwert als auch die Schrittweite aus Variablen, Ausdrücken oder Konstanten bestehen. Ein Beispiel:

```
10 A = 20 : B = 10  : C = -.5
20 FOR I = A TO B STEP C
30 PRINT I * 3 + 5
40 NEXT
```

A steht hier für den Anfangswert, B für den Endwert und C für die Schrittweite.

Es besteht auch die Möglichkeit, mehrere FOR...NEXT-Schleifen zu verschachteln. Dabei ist jedoch darauf zu achten, daß die inneren Schleifen in sich abgeschlossen sind. Hier ein Beispiel:

```
10 FOR I = 1 TO 10
20 FOR J = 0 TO 20 STEP 2
30 FOR K = 7 TO 50
40 .... (Anweisungen)
50 NEXT K
60 NEXT J
70 NEXT I
```

Beachten Sie, daß in diesem Beispiel die innere Schleife die Laufvariable K, die mittlere J und die äußere I hat. Falsch dagegen ist es, die Reihenfolge von NEXT zu vertauschen, wie im folgenden Beispiel (hier nur Zeilen 50 bis 70):

```
50 NEXT I
60 NEXT K    (f a l s c h !)
70 NEXT J
```

Bevor wir die FOR...NEXT-Anweisung verlassen, hier noch ein praktisches Beispiel zur Zinsberechnung:

```
10 REM ZINSBERECHNUNG
20 INPUT"KAPITAL"; K
30 INPUT"ZINSSATZ (PROZENT)"; ZP
40 INPUT"LAUFZEIT (JAHRE)"; L
50 Z = ZP/ 100
60 B = K
70 FOR I = 1 TO L
80 B = B + B * Z
90 NEXT
100 PRINT"DER BETRAG IST"; B
```

In Zeile 20 wird per INPUT-Anweisung das Kapital eingegeben und der Variablen K zugeordnet. Das gleiche geschieht in Zeile 30 mit dem Zinssatz in Prozent (ZP) und in Zeile 40 mit der Laufzeit in Jahren (L). Zeile 50 rechnet den Zinssatz in Prozent (ZP) in einen Dezimalbruch (Z) um. In Zeile 60 steht die Variable B für den jeweiligen Betrag, der zunächst mit dem Kapital K gleichgesetzt wird. Die Zeilen 70 bis 90 bilden dann eine Schleife mit einer FOR...NEXT-Anweisung. Für jedes Jahr der Laufzeit L wird hier der jeweilige Betrag berechnet. Zeile 100 gibt schließlich das Endergebnis auf dem Bildschirm aus. Beachten Sie bei dieser PRINT-Anweisung, daß sie zunächst den String "DER BETRAG IST" und dann den Inhalt der Fließkommavariablen B ausgibt.

Wir lassen das Programm jetzt einmal laufen, wobei das Kapital von 1000000 über 10 Jahre mit einem Zinssatz von 12% verzinst werden soll:

```
KAPITAL? 1000000
ZINSSATZ (PROZENT)? 12
LAUFZEIT (JAHRE)? 10
DER BETRAG IST 3105848.21

READY.
```

Das Ergebnis wäre sicherlich übersichtlicher und besser zu lesen, wenn es so geschrieben würde: 3 105 848.21. Im weiteren Verlauf dieses Buches werden wir noch die PRINT USING-Anweisung kennen lernen, die eine formatierte Ausgabe ermöglicht.

Ändern wir dieses Programm leicht ab, so können wir es zur Berechnung von Annuitäten verwenden. Der einzige Unterschied besteht darin, daß zu Beginn eines jeden Jahres ein neuer Betrag hinzukommt. Deshalb erhöht sich der Wert von B nicht nur um die Zinsen, sondern auch um die jeweils neue Einlage.

Diese Einlage ist in der Variablen E abgelegt. Der Betrag B wird dann in zwei Stufen berechnet, indem zunächst die jährliche Einlage hinzugezählt und schließlich aus der Summe die Zinsen des letzten Jahres berechnet und hinzuaddiert werden:

```
10 REM ANNUITAETEN
20 INPUT"ANFANGSBETRAG"; B
30 INPUT"ZINSSATZ (PROZENT)"; ZP
40 INPUT"EINLAGE PRO JAHR"; E
50 INPUT"LAUFZEIT (JAHRE)"; L
60 Z = ZP/ 100
70 FOR I = 1 TO L
80 B = B + E
90 B = B + B * Z
100 NEXT
110 PRINT"DER BETRAG IST"; B
```

Nachdem das Programm mit RUN gestartet wurde, testen wir es mit folgenden Werten:

```
ANFANGSBETRAG? 1000000
ZINSSATZ (PROZENT)? 15
EINLAGE PRO JAHR? 10000
LAUFZEIT (JAHRE)? 10
DER BETRAG IST 4279050.5
```

Bitte beachten Sie, daß im Anfangsbeitrag die erste Jahreseinlage nicht enthalten ist, weshalb sich der Betrag zunächst aus dem Anfangsbetrag und der ersten Jahreseinlage zusammensetzt.

3.2 DO...LOOP

Die FOR...NEXT-Anweisung teilt dem Computer mit, wie oft er die zwischen FOR und NEXT liegenden Anweisungen ausführen soll. Dies ist in der Laufvariablen hinter FOR angegeben. Es gibt jedoch Fälle, in denen es von vornherein nicht bekannt ist, wie oft eine Schleife auszuführen ist. Dies geschieht dann solange, bis eine bestimmte Bedingung erfüllt ist. Das BASIC 7.0 bietet hierzu die DO...LOOP-Anweisung.

Ähnlich wie bei FOR...NEXT umfaßt die Schleife hier sämtliche zwischen DO und LOOP liegenden Anweisungen; dabei ist jedoch keine vorgegebene Anzahl von Durchläufen angegeben. Sie wird unendlich oft ausgeführt, es sei denn, sie enthält eine Bedingung, die zum Abbruch führt.

Um uns dies zu verdeutlichen, wollen wir nochmals alle Zahlen zwischen 1 und 10 ausgeben, diesmal aber nicht mit der FOR...NEXT-, sondern mit der DO...LOOP-Anweisung:

```
10 I = 1
20 DO
30 IF I > 10 THEN EXIT
40 PRINT I;
50 I = I + 1
60 LOOP
```

Nach Eingabe von RUN erfolgt

```
1 2 3 4 5 6 7 8 9 10

READY.
```

Als Zählvariable verwenden wir hier wieder I. In Zeile 10 erhält sie den Anfangswert 1. Die DO...LOOP-Schleife umfaßt die Zeilen 20 bis 60, wobei Zeile 40 den jeweiligen Wert von I auf dem Bildschirm ausgibt. Zeile 50 erhöht den Wert von I jeweils um 1, d.h. bei jedem Schleifendurchlauf wird I fortgezählt.

Dieser Vorgang würde sich nun unendlich oft wiederholen, hätten wir nicht die Abbruchbedingung in Zeile 30 eingebaut. Versuchen Sie einmal, Zeile 30 zu löschen und das Programm erneut zu starten. Es führt dann eine Endlosschleife aus, die nur durch Drücken der STOP-Taste abgebrochen werden kann.

Mit der Abbruchbedingung in Zeile 30 wollen wir uns jetzt näher befassen. Sie lautet in allgemeiner Form

IF (Bedingung) THEN EXIT

Ist die Bedingung erfüllt, wird die Wiederholung abgebrochen, anderenfalls wird mit der nächsten Anweisung fortgefahren.

In unserem speziellen Fall ist die Bedingung erfüllt, wenn I größer als 10 ist, denn wir wollen ja nur die Zahlen bis ein schließlich 10 ausgeben. Hier nochmals Zeile 30:

```
30 IF I > 10 THEN EXIT
```

Das Zeichen ">" ist ein Vergleichssymbol, das wir bereits aus der Algebra kennen. Dabei wird der jeweilige Wert von I mit der Zahl 10 verglichen und zwar solange, bis er größer als 10 ist.

Insgesamt gibt es in BASIC sechs Vergleichssymbole, die in einer IF-Bedingung auftreten können. Sie dienen zum Vergleichen von konstanten Werten, Variablen und Ausdrücken. Nachfolgend eine tabellarische Zusammenfassung dieser Symbole:

Vergleichssymbol	**Bedeutung**
=	gleich
<	kleiner als
<=	kleiner gleich
>	größer als
>=	größer gleich
<>	ungleich

Auf keinen Fall darf aber das Gleichheitszeichen als Vergleichssymbol mit demjenigen in einer Zuordnungsanweisung verwechselt werden, da es sich hier um völlig verschiedene Anwendungen handelt. In einer Zuordnung sagt

```
Y = Z
```

aus, daß der Wert der Fließkommavariablen Y gleich demjenigen der Variablen Z gesetzt wird. Schreiben wir dagegen

```
IF Y = Z THEN EXIT
```

so liegt eine Bedingung vor, die erst erfüllt ist, wenn Y den gleichen Wert wie Z erreicht hat.

Grundsätzlich sind die Vergleichssymbole auch für Strings anwend bar, wobei allerdings die alphabetische Reihenfolge geprüft wird.

Wie wir bereits wissen, setzen sich Strings aus sogenannten ASCII-Zeichen zusammen, wobei jedem Buchstaben intern ein bestimmter Zahlenwert zugeordnet ist. So entspricht der Buchstabe A dem Code 65, B dem Code 66 usw. und Z dem Code 90. Dies können wir anhand der folgenden beiden Anweisungen überprüfen:

```
PRINT CHR$(65)
A

READY.

PRINT ASC("A")
65

READY.
```

Die CHR$-Anweisung erzeugt für einen vorgegebenen ASCII-Code das entsprechende Zeichen; in unserem Beispiel erzeugt die Zahl 65 den Buchstaben A. ASC ist hiervon die Umkehrfunktion, die in unserem Fall den dem Buchstaben A entsprechenden ASCII-Code erzeugt.

Normalerweise sind die CHR$- und ASC-Anweisungen nur für den fortgeschrittenen Programmierer interessant, der sie meist nur für ganz spezielle Anwendungsbereiche benötigt. Für uns sollen sie im Augenblick ausschließlich zum Verständnis dienen, warum der Computer auch Strings mit den Vergleichssymbolen in alphabetischer Reihenfolge vergleichen kann. Gegenüber den Zahlenvergleichen haben die Symbole hier eine etwas andere Bedeutung, wie die nachfolgende Aufstellung zeigt:

Vergleichssymbol	Bedeutung
=	gleich
<	kleiner in alphabetischer Reihenfolge
<=	kleiner oder gleich in alphabetischer Folge
>	größer in alphabetischer Reihenfolge
>=	größer oder gleich in alphabetischer Folge
<>	ungleich

Hier noch einige Beispiele zum Stringvergleich:

HANS < MARTIN
UNGER > MEYER
SCHUH < SCHUHLADEN
DREIMAL > DREI MAL

Beachten Sie, daß beim vorletzten Beispiel das Wort SCHUH kleiner als SCHUHLADEN ist. Letzteres beginnt zwar vorne mit dem Wort SCHUH, an das jedoch noch zusätzliche Zeichen angefügt sind.

Auch das Wort DREIMAL ist größer als DREI MAL, da DREI MAL ein Leerzeichen (ASCII-Code 32) enthält, das kleiner ist als die Buchstaben des Alphabets.

Für die DO...LOOP-Anweisung gibt es zwei Sonderformen, die ein noch komfortableres Programmieren ermöglichen. Es handelt sich dabei um

DO WHILE ... LOOP

und

DO UNTIL ... LOOP

DO WHILE bedeutet, daß der Computer eine Reihe von Anweisungen solange auszuführt, solange eine Bedingung wahr ist; bei DO UNTIL dagegen werden die Anweisungen ausgeführt, bis eine Bedingung wahr ist.

Als Anwendungsbeispiel dieser beiden Anweisungsformen verwenden wir wieder unsere Aufgabe, bei der die Zahlen zwischen 1 und 10 auf dem Bildschirm ausgegeben werden:

```
10 I = 1
20 DO WHILE I <= 10
30 PRINT I;
40 I = I + 1
50 LOOP
```

Im vorliegenden Fall wird der Inhalt von I solange auf dem Bildschirm ausgegeben und anschließend um 1 erhöht, wie I kleiner oder gleich 10 ist. Dieselbe Aufgabe läßt sich auch mit DO UNTIL programmieren:

```
10 I = 1
20 DO UNTIL I > 10
30 PRINT I;
40 I = I + 1
50 LOOP
```

Hier werden die Anweisungen solange ausgeführt, bis der Inhalt von I größer als 10 ist. Für welche der beiden Formen Sie sich entscheiden, ist eine Frage des Geschmacks und der Zweckmäßigkeit. Im vorliegenden Fall ist der Programmieraufwand in etwa der gleiche.

3.3 READ und DATA

Wir kennen bereits die INPUT-Anweisung, die während des Programmablaufs Daten über die Tastatur vom Bediener anfordert. Es gibt aber noch eine weitere Möglichkeit, Daten ins Programm zu übernehmen. Solche Daten sind in DATA-Anweisungen abgelegt und werden bei Bedarf zur weiteren Verarbeitung mit der READ-Anweisung gelesen.

Hier kommen in erster Linie konstante Datenwerte in Frage und nicht variable wie bei der INPUT-Anweisung. Es kommt nämlich häufig vor,

daß ein Programm mit festen Tabellen arbeiten muß, die dann aus DATA-Zeilen eingelesen werden können. Anderenfalls müßte sie der Bediener jeweils von Hand eingeben oder von einem externen Datenträger (Diskette oder Band) einladen.

Die letzte Möglichkeit empfiehlt sich nur bei extrem umfangreichen Datendateien, die, wären sie in DATA-Zeilen abgelegt, zuviel Speicherplatz beanspruchen würden. Auch eine sich ständig ändernde Datei (z.B. Adressendatei) legt man besser auf einem Datenträger ab. Halten sich dagegen die Datenmengen in kleinerem Rahmen und bleiben sie bei jedem Programmablauf gleich, ist es oft sinnvoll, sie in DATA-Anweisungen abzulegen, die ein fester Bestandteil des BASIC-Programmtextes sind.

Bereits in Kapitel 2 haben wir mit Hilfe der PRINT-Anweisung eine Tabelle auf dem Bildschirm ausgegeben. Diese Tabelle, die den Namen, die Punktzahl und die Noten von Klassenarbeiten für verschiedene Schüler enthält, wollen wir hier zu Demonstrationszwecken mit READ- und DATA-Anweisungen ausgeben. Dazu dient das folgende Programm:

```
10 PRINT "NAME","PUNKTE","NOTE"
20 PRINT
30 DO
40 READ NA$ : IF NA$  = "ENDE" THEN EXIT
50 READ P, N
60 PRINT NA$, P, N
70 LOOP
100 DATA HANS, 35, 3
110 DATA NICOLE, 40, 4
130 DATA STEFAN, 50, 2.7
140 DATA CHRISTIAN, 38, 4.3
150 DATA URSULA, 60, 2
160 DATA ENDE
```

Zu Beginn erscheint wieder die Kopfzeile und eine Leerzeile (Zeilen 10 und 20). Die Zeilen 30 bis 70 enthalten eine DO...LOOP-Schleife und die Zeilen 100 bis 160 die eigentliche "Datei", die aus DATA-Zeilen besteht.

In einer DATA-Anweisung können eine oder mehrere Daten stehen. Sind mehrere Daten vorhanden, wie in unserem Fall, müssen sie durch Kommas getrennt sein.

Beachten Sie bitte auch, daß Strings in DATA-Zeilen nicht in Anführungszeichen eingeschlossen sein brauchen. Es ist allerdings kein Fehler, dies zu tun. Enthält ein String ein Komma, so ist dies jedoch zwingend notwendig, da Kommas ansonsten als Trennzeichen zwischen den verschiedenen Daten dienen. Das folgende Beispiel verdeutlicht dies:

```
100 DATA "BIS GLEICH, AUF WIEDERSEHEN", TSCHUESS,
"ADE"
```

Diese DATA-Zeile enthält drei verschiedene Stringdaten. Wäre "BIS GLEICH, AUF WIEDERSEHEN" nicht in Anführungszeichen eingeschlossen, würde BASIC annehmen, daß hier zwei Strings, also BIS GLEICH und AUF WIEDERSEHEN vorliegen. TSCHUESS dagegen gilt als ein Wert und braucht nicht in Anführungszeichen zu stehen. Das gleiche gilt für ADE, wo die Anführungszeichen keine Wirkung haben und durchaus weggelassen werden könnten.

Doch nun zurück zu unserem Programm. In der Schleife liest die READ-Anweisung in Zeile 40 zunächst den ersten Datenwert, der hier ein Name (String) ist, aus der Datentabelle und ordnet ihn der Stringvariablen NA$ zu. Anschließend folgt die Abfrage nach dem Label "Ende". Ein Label ist in der Computertechnik eine Kennzeichnung, die zu bestimmten Abfragen dient. In unserem Fall haben wir den String "ENDE" an das Tabellenende gesetzt. Wird nun dieses Wort (Label) als Name eingelesen, weiß der Computer, daß die Tabelle abgearbeitet ist und verläßt die Schleife aufgrund von EXIT. Statt ENDE könnten wir auch jedes andere beliebige Wort als Abbruchkriterium (Label) vereinbaren.

Enthält nun NA$ einen anderen Wert als "ENDE", ist also das Tabellenende noch nicht erreicht, wird in einer weiteren READ-Anweisung noch die Punktzahl und die Note des Schülers eingelesen. Genauso wie bei DATA können auch hinter READ mehrere durch Komma getrennte Variablennamen stehen. Zeile 60 gibt dann sämtliche Werte auf dem Bildschirm aus, worauf das Programm in Zeile 70 an den Schleifenanfang zurückspringt.

Nun noch ein paar Worte zum Aufbau von Programmen mit DATA-Zeilen. Findet BASIC nach dem Programmstart durch RUN erstmals eine READ-Anweisung vor, so liest es den ersten Wert in der ersten DATA-Zeile. Wird READ ein zweites Mal vorgefunden, so ist der zweite Datenwert an der Reihe usw. BASIC verfügt nämlich über einen internen Zeiger und weiß deshalb, welche DATA-Anweisung als nächste an der Reihe ist. Es spielt deshalb keine Rolle, an welcher Stelle die DATA-Zeilen im Programm ste-

hen. Entscheidend ist nur ihre Reihenfolge im Programmtext. Im vorliegenden Beispiel hätten wir sie genauso gut auch an den Programmanfang setzen oder zwischen die anderen BASIC-Zeilen verteilen können, was allerdings von der Übersichtlichkeit her nicht zu empfehlen ist.

Beim Programmstart durch RUN wird der interne DATA-Zeiger automatisch auf den ersten Datenwert gesetzt. Manchmal ist es jedoch erforderlich, ihn während des Programmablaufs wieder an den Anfang zurückzusetzen. Hierzu dient die Anweisung

```
RESTORE
```

BASIC 7.0 bietet darüberhinaus noch die Möglichkeit, den Zeiger auf jede beliebige DATA-Zeile zu setzen, wodurch die Datenwerte erst ab der angegebenen Zeile gelesen werden. Dies wollen wir anhand unseres Programms einmal ausprobieren. Dazu setzen wir folgende Zeile an den Programmanfang:

```
5 RESTORE 130
```

Wenn Sie jetzt das Programm starten, werden als erstes die in Zeile 130 abgelegten Daten ausgegeben.

Wird die READ-Anweisung öfter aufgerufen als Datenwerte vorhanden sind, läuft also der interne DATA-Zeiger über den letzten Datenwert hinaus, erscheint die Fehlermeldung

```
?OUT OF DATA ERROR IN (Zeilennummer)
```

und führt zum Programmabbruch.

3.4 IF...THEN...ELSE

Die IF-Anweisung haben wir bereits bei der Schleifenprogrammierung kennengelernt, wo sie zur Bestimmung des Abbruchkriteriums diente. Sie hatte dort die allgemeine Form

IF (Bedingung) THEN EXIT

Aber auch jede andere Abfrage kann durch IF vorgenommen werden. Stößt nämlich der Computer auf eine IF-Anweisung, prüft er, ob die Bedingung wahr oder falsch ist. Ist die Bedingung wahr, werden die hinter THEN stehenden Anweisungen ausgeführt, anderenfalls werden sie übersprungen. Betrachten wir diesen Vorgang anhand eines einfachen Beispiels, in dem die Zahl 25 erraten werden soll:

```
5 REM ZAHLENRATEN
10 INPUT "BITTE ZAHL EINGEBEN"; Z
20 IF Z = 25 THEN PRINT "RICHTIG!" : GOTO 40
30 PRINT "FALSCH!"
40 PRINT "WEITERRATEN - BITTE LEERTASTE DRUECKEN"
50 PRINT "WENN ANDERE TASTE - ENDE"
60 GETKEY A$ : IF A$ = " " THEN 10
70 END
```

In Zeile 10 ist eine beliebige Zahl Z einzugeben. In Zeile 20 prüft dann die IF-Anweisung, ob es sich hierbei um die richtige Zahl 25 handelt. Ist dies der Fall, werden die hinter THEN stehenden Anweisungen ausgeführt, d.h., es erscheint das Wort "RICHTIG!" auf dem Bildschirm. Damit das Programm nun nicht mit Zeile 30 fortfährt, steht dahinter noch die Sprunganweisung GOTO 40, d.h., die Programmausführung wird in Zeile 40 fortgesetzt.

Ist die Zahl Z eine andere als 25, werden die hinter der IF...THEN stehenden Anweisungen nicht ausgeführt und mit Zeile 30 fortgefahren.

In beiden Fällen wird der Programmteil ab Zeile 40 ausgeführt, wo zunächst zwei Strings zur Information auf dem Bildschirm ausgegeben werden. In Zeile 60 steht die Anweisung GETKEY A$. Hier hängt das Programm solange in einer Warteschleife, bis der Bediener eine beliebige Taste drückt. Ist dies geschehen, fragt eine weitere IF-Anweisung, ob es sich hier bei um die Leertaste handelt, gekennzeichnet durch einen Leerstring von

einem Zeichen Länge. Ist die Bedingung erfüllt, findet ein Sprung zur INPUT-Anweisung in Zeile 10 statt, wo eine neue Zahl einzugeben ist. Ist eine andere Taste gedrückt, also die Bedingung nicht erfüllt, wird in Zeile 70 die END-Anweisung ausgeführt.

In diesem Fall könnte man die END-Anweisung auch fortlassen, da der Programmtext ohnehin an dieser Stelle zu Ende ist. END ist überall da erforderlich, wo die Programmausführung innerhalb des Textes beendet wird, weil sie andererseits mit den unteren Zeilen fortfahren würde.

Unser Zahlenrateprogramm können wir dank des leistungsstarken BASIC 7.0 auch noch etwas kompakter schreiben, wie das folgende Listing zeigt:

```
5 REM ZAHLENRATEN
10 INPUT "BITTE ZAHL EINGEBEN"; Z
20 IF Z = 25 THEN PRINT "RICHTIG!": ELSE PRINT "FALSCH!"
30 PRINT "WEITERRATEN - BITTE LEERTASTE DRUECKEN"
40 PRINT "WENN ANDERE TASTE - ENDE"
50 GETKEY A$ : IF A$ = " " THEN 10: ELSE END
```

Hier werden die beiden Möglichkeiten einer richtigen bzw. falschen Zahleneingabe in einer Zeile abgehandelt. Ist die Bedingung erfüllt, wird wie zuvor "RICHTIG!" ausgegeben; ist sie nicht erfüllt, werden stattdessen die hinter ELSE stehenden Anweisungen ausgeführt. Dasselbe wiederholt sich in Zeile 50 mit der Abfrage der Leertaste.

Wichtig! Die Anweisungen IF...THEN und IF...THEN...ELSE müssen einschließlich der Bedingung und sämtlicher in diesem Zusammenhang auszuführenden Anweisungen in einer Programmzeile stehen. Ausnahmen sind nur im Zusammenhang mit BEGIN...BEND (siehe unten) zulässig. Steht hinter THEN bzw. ELSE eine Zeilennummer, so findet ein Sprung in die betreffende Zeile statt. Man könnte hierfür auch THEN GOTO (Zeile) bzw. ELSE GOTO (Zeile) schreiben.

3.5 BEGIN....BEND

Wie wir im letzten Absatz gesehen haben, darf sich eine komplette IF...THEN- bzw. IF...THEN...ELSE-Anweisung nur über eine Programmzeile erstrecken, die maximal 160 Textzeichen einschließlich der Zeilennummer umfaßt. Diese Einschränkung ist allerdings zu umgehen, indem hinter dem THEN- bzw. ELSE-Teil beliebig viele Anweisungen in eine

BEGIN...BEND Umklammerung eingeschlossen werden können, die beliebig viele Programmzeilen umfassen kann. Somit lassen sich auch größere Programmblöcke ohne GOTO programmieren, was nicht nur die Übersichtlichkeit, sondern auch die Ablaufgeschwindigkeit des Programms erhöht. Bei jedem GOTO-Befehl muß der BASIC-Interpreter zunächst die Zeilennummer aus dem Dezimal- ins Binärformat umrechnen und anschließend die betreffende Zeile suchen.

Bei der Anwendung der BEGIN...BEND-Anweisung treten keine Zeilennummern mehr auf. Dieser Vorteil macht sich besonders dann bemerkbar, wenn der betreffende Programmteil in einen anderen Zeilenbereich übernommen werden soll. Außerdem entfällt das sonst lästige Berechnen der Zeilennummern bei GOTO-Befehlen.

Die BEGIN...BEND-Anweisung wollen wir uns anhand eines Beispiels verdeutlichen:

```
10 PRINT "HABEN WIR HEUTE SCHOENES WETTER (J/N) ?"
20 GETKEY A$
30 IF A$ = "J" THEN BEGIN
40 PRINT "DANN HABEN WIR HEUTE EINEN HERRLICHEN
TAG"
50 PRINT "UND WOLLEN UNS EIN PAAR SCHOENE STUNDEN"
60 PRINT "IM FREIBAD GOENNEN"
70 BEND : ELSE BEGIN
80 PRINT "SCHADE, HEUTE REGNET ES"
90 PRINT "ABER WIR LASSEN UNS DIE LAUNE NICHT"
100 PRINT "VERDERBEN UND SPIELEN MONOPOLY"
110 BEND
```

Dieses Programm fragt Sie, wie heute das Wetter ist. Antworten Sie mit "J", werden die Zeilen 40 bis 60 ausgeführt, die Ihnen raten, ins Freibad zu gehen. Geben Sie "N" oder irgend eine andere Taste ein, werden die Zeilen 80 bis 100 ausgeführt, die den Hinweis geben, bei schlechtem Wetter Monopoly zu spielen. Ausgeführt werden jeweils die Zeilen, die zwischen BEGIN und BEND stehen.

Beachten Sie bitte Zeile 70. BEND des THEN-Teils und ELSE BEGIN stehen hier in einer Zeile hintereinander. Wenn Sie diese Schreibweise verletzen, arbeitet das Programm nicht ordnungsgemäß.

3.6 Die Booleschen Operatoren

Haben wir z.B. die Bedingung

X = Y

so sagt sie aus, daß der Wert von X gleich dem Wert von Y ist. Darüber hinaus ist es wünschenswert, auch Bedingungen in zusammengesetzten Sätzen zu schreiben, wie z.B. "Der Wert von U ist gleich dem Wert von V und der Wert von X ist kleiner als der Wert von Y. Durch Einsetzen der entsprechenden logischen Operatoren AND, OR und NOT (deutsch und, oder, nicht), können Bedingungen gemäß dem oben zusammengesetzten Satz geschrieben werden:

U = V AND X < Y

Bevor wir uns nun mit den einzelnen logischen Operatoren befassen, wollen wir uns darüber klar werden, wie der Computer reagiert, wenn eine Bedingung wahr oder falsch ist. In einigen Programmiersprachen gibt es dafür einen speziellen Datentyp, der nur die beiden Werte TRUE (wahr) und FALSE (falsch) annehmen kann. Ein solcher Datentyp existiert in BASIC nicht, stattdessen findet der Datentyp Integer Verwendung. 0 steht hier für falsch und -1 für wahr. Bedingungen wie A = B sind Ausdrücke, die den Wert -1 ergeben, wenn die Bedingung wahr ist und 0, wenn sie falsch ist:

```
PRINT 2 = 2
-1

READY.

PRINT 12 > 20
 0

READY.
```

```
PRINT "HANS" < "MARTIN"
-1

READY.

PRINT "FRITZ" = "FRITZCHEN"
 0

READY.
```

Ergibt eine durch IF geprüfte Bedingung den Wert -1, dann werden die Anweisungen im THEN-Teil ausgeführt, entsteht dagegen der Wert 0, dann sind es die Anweisungen im ELSE-Teil.

```
IF -1 THEN PRINT "WAHR" ELSE PRINT "FALSCH"
WAHR

READY.

IF 0 THEN PRINT "WAHR" ELSE PRINT "FALSCH"
FALSCH

READY.
```

Die IF-Anweisung akzeptiert in der Regel auch andere Werte (außer 0), um "wahr" zu erzeugen, wie im folgenden Beispiel:

```
IF 3 THEN PRINT "WAHR" : ELSE PRINT "FALSCH"
WAHR

READY.
```

BASIC kennt somit nur einen Wert, nämlich den Wert 0, der als "falsch" gedeutet wird, alle anderen Werte bedeuten "wahr". Obwohl alle Werte, die ungleich 0 sind, den Zustand "wahr" darstellen, wollen wir hier trotzdem den Wert -1 weiterverwenden, da er auch von den Booleschen Operatoren erzeugt wird. Im folgenden wollen wir uns mit den Operatoren einzeln befassen.

3.6.1 NOT

Ist einer Bedingung NOT vorangestellt, wird sie verneint, ebenso wie auch ein Satz, der das Wort "nicht" enthält, verneint wird.

```
NOT 7 < 5
```

entspricht somit der Aussage, daß 7 nicht kleiner als 5 sei. Um dies zu erreichen, muß nämlich NOT von -1 nach 0 und von 0 nach 1 umgeändert werden:

```
PRINT NOT -1; NOT 0
 0 -1

READY.

PRINT NOT 7 < 5
-1

READY.
```

3.6.2 AND

Betrachten wir zunächst die Bedingung:

```
2 < 3 AND 10 > 12
```

In Worten ausgedrückt bedeutet dies: "2 ist kleiner als 3 und 10 ist größer als 12". Ein zusammengesetzter Satz mit AND ist nur wahr, wenn beide Teilsätze wahr sind. AND erzeugt den Wert -1 also nur dann, wenn beide betrachteten Werte wahr, also auch -1 sind.

```
PRINT 0 AND 0; 0 AND -1; -1 AND 0; -1 AND -1
 0  0  0 -1

READY.

PRINT 2 < 3 AND 10 > 12
 0

READY.
```

3.6.3 OR

Zunächst betrachten wir wieder die Bedingung:

```
2 < 3 OR 10 > 12
```

Sie sagt aus, daß 2 kleiner als 3 oder 10 größer als 12 ist oder daß beides zutrifft. Durch OR entsteht der Wert -1 (wahr), wenn einer oder beide Werte -1 (wahr) sind.

```
PRINT 0 OR 0; 0 OR -1; -1 OR 0; -1 OR -1
 0 -1 -1 -1

READY.

PRINT 2 < 3 OR 10 > 12
-1

READY.
```

3.6.4 XOR

Die Bedingung

```
2 < 3 XOR 10 > 12
```

sagt aus, daß 2 kleiner als 3 und 10 größer als 12 ist, aber daß nicht beides zutrifft. Nur wenn beide Werte gleich sind, unterscheiden sich OR und XOR, wobei OR den Wert -1 und XOR den Wert 0 erzeugt:

```
PRINT 0 XOR 0; 0 XOR -1; -1 XOR 0; -1 XOR -1
 0 -1 -1  0

READY.
```

Abschließend wollen wir noch einen praktischen Anwendungsfall für die Booleschen Operatoren betrachten. In der Geodäsie ist es üblich, zum Zwecke weiterer Fernbeobachtungen eine Strecke von 50 m genau einzumessen. Dazu wird die Strecke mehrfach gemessen, um sicherzustellen, daß sich der Abstand der Endpunkte innerhalb der zulässigen Toleranzgrenze von 0,03 m bewegt. Messungen, die über diese Toleranzgrenze hinausgehen, werden nicht weiter verwertet und scheiden aus.

```
10 REM TOLERANZMESSUNG EINER MESSTRECKE
20 PRINT "LAENGE", "ANGENOMMEN"
30 PRINT
40 READ L
50 DO
60 IF L = 9999 THEN EXIT
70 PRINT L,
80 IF L>=49.97 AND L<=50.03 THEN PRINT "JA": ELSE PRINT
"NEIN"
90 READ L
100 LOOP
110 DATA 50.01, 50.07, 49.97, 49.99, 49.93, 9999
```

Nach Eingabe von RUN erscheint folgende Auswertung:

```
LAENGE       ANGENOMMEN

 50.01          JA
 50.07          NEIN
 49.97          JA
 49.99          JA
 49.93          NEIN
```

READY.

Die einzelnen Meßwerte L werden hier aus DATA-Anweisungen eingelesen, und in einer DO...LOOP-Schleife getestet und ausgegeben. Ist ihre Länge innerhalb der zulässigen Toleranzgrenze, wird zusätzlich das Wort "JA" ausgegeben, anderenfalls das Wort "NEIN". Das Tabellenende ist hier wieder mit einem Label gekennzeichnet, denn wenn das Programm den Wert 9999 erkennt, bricht es ab.

4 Programmaufbau und Fehlerbehandlung

In diesem Kapitel wollen wir uns mit den wesentlichen Elementen zum Aufbau von BASIC-Programmen befassen und in diesem Zusammenhang weitere wichtige Anweisungen kennenlernen. Zwar wissen wir bereits, daß wir Programmzeilen beliebig aneinanderreihen können, um sie dann nacheinander abzuarbeiten, wobei wir zweifellos ein ablauffähiges Programm erhalten. Es bleibt jedoch die Frage offen, ob das Programm optimal aufgebaut ist, insbesondere was den Programmieraufwand, die Rechenzeit und die Übersichtlichkeit betrifft. Wir werden uns deshalb nunmehr mit Funktionen und Unterprogrammen beschäftigen, aus denen sich gut strukturierte Programme zusammensetzen. Ein auf diese Weise sinnvoll durchgeführter Programmaufbau ist der beste Weg, um komplizierte Aufgabenstellungen in übersichtliche Teile zu untergliedern.

Je umfangreicher ein Programm wird, desto größer ist auch die Wahrscheinlichkeit, daß Programmierfehler auftreten. Aus diesem Grunde befassen wir uns am Schluß des Kapitels mit Möglichkeiten, Fehler aufzuspüren und zu beheben.

4.1 Eingebaute Funktionen

Der BASIC-Interpreter enthält einige eingebaute Funktionen, die uns das Programmieren erleichtern. Sie dienen zur Definition von Zahlen- oder Stringoperationen und können als separate Programmteile betrachtet werden, die nicht gesondert definiert werden müssen.

Nun gibt es im BASIC 7.0 einfachere und kompliziertere Funktionen. Die wissenschaftlichen Funktionen, die wir schon in Kapitel 2 kennengelernt haben, gehören zu den komplizierteren. Zweifellos könnten Sie z.B. die Si-

nusfunktion auch selbst programmieren, indem Sie ein Programm schreiben, das bei jedem Aufruf eine Potenzreihe durchrechnet. Dazu brauchten Sie allerdings Kenntnisse in der Potenzreihenentwicklung und müßten darüber hinaus beachten, daß diese Reihe aureichend schnell konvergiert, um bei einer minimalen Anzahl von Schleifendurchläufen die größtmögliche Rechengenauigkeit zu erhalten.

Diese und ähnliche Aufgaben nimmt Ihnen nun der BASIC-Interpreter ab, indem er eine interne Maschinenroutine aufruft, welche die Berechnungen auch noch um ein Vielfaches schneller ausführt, als es mit einem BASIC-Programm möglich wäre.

4.1.1 ABS

Die ABS-Funktion bestimmt den absoluten Wert einer Zahl, d.h. sie führt eine Operation aus, bei der negative Zahlen ein positives Vorzeichen erhalten, während es bei positiven Zahlen gleich bleibt. In anderen Worten: Es spielt keine Rolle, ob der Ausgangswert positiv oder negativ ist, das Ergebnis ist auf jeden Fall positiv.

Betrachten wir dies an einem Beispiel: Der Wert -3 wird nach Ausführung der ABS-Funktion zu +3 oder einfach zu 3, während der Wert 100 gleich bleibt. In BASIC sieht das folgendermaßen aus:

```
PRINT ABS(-3)
 3

READY.


PRINT ABS(100)
 100

READY.
```

Hinter ABS muß das Argument, wie bei allen Funktionen, in Klammern eingeschlossen sein. Dabei muß es nicht unbedingt ein konstanter Zahlenwert sein, sondern kann auch aus einer Variablen oder einem arithmetischen Ausdruck bestehen wie in den folgenden Beispielen:

```
A = -4 : PRINT ABS(A)
 4

READY.

PRINT ABS (100*(5-2))
 300

READY.
```

Ein typischer Anwendungsfall für die ABS-Funktion ist die Bestimmung von Abweichungen in der Toleranzmessung. Betrachten wir nochmals unser Beispiel aus Kapitel 3, in dem die Meßwerte einer Meßstrecke ausgewertet wurden. Wir wollen es hier etwas erweitern, indem wir nicht nur angeben wollen, ob die Messung verwertbar ist oder nicht, sondern zusätzlich noch den absoluten Fehler in Metern. Dabei interessiert es bei solchen Messungen nicht, ob der Sollwert nach oben oder nach unten abweicht, d.h., ob die Differenz positiv oder negativ ist, denn man verwertet in solchen Fällen immer den Absolutwert.

```
10 REM TOLERANZMESSUNG EINER MESSTRECKE
20 PRINT "LAENGE","FEHLER","ANGENOMMEN"
30 PRINT
40 READ L
50 DO
60 IF L = 9999 THEN EXIT
70 PRINT L, ABS(50-L),
80 IF L>=49.97 AND L<=50.03 THEN PRINT "JA": ELSE PRINT
"NEIN"
90 READ L
100 LOOP
110 DATA 50.01, 50.07, 49.97, 49.99, 49.93, 9999
```

Beachten Sie Zeile 70, in der außer der gemessenen Länge auch noch der absolute Fehler ausgegeben wird. Nach Eingabe von RUN erscheint die Auswertung:

```
LAENGE        FEHLER        ANGENOMMEN

  50.01        .01          JA
  50.07        .07          NEIN
  49.97        .03          JA
  49.99        .01          JA
  49.93        .07          NEIN

 READY.
```

4.1.2 INT

Viele Zahlen bestehen aus einem ganzzahligen und einem gebrochenen Teil. So setzt sich z.B. die Zahl 123.456 aus 123 als ganzzahligem und .456 als gebrochenem Teil zusammen.

Die INT-Funktion trennt nun den gebrochenen Teil ab und gibt den ganzzahligen Teil wieder. Dabei handelt es sich strenggenommen um den nächstkleineren ganzzahligen Wert, was besonders bei negativen Zahlen zu Irrtümern führen kann. Hier ein paar Beispiele:

```
PRINT INT(123.456)
 123

READY.

PRINT INT(2.2)
 2

READY.

PRINT INT(3.5)
 3

READY.
```

```
PRINT INT(-3.5)
-4

READY

PRINT INT(-5)
-5

READY.
```

Beachten Sie besonders die letzten beiden Beispiele. Von -3.5 ist die nächstkleinere ganze Zahl -4 und nicht -3. -5 hat keinen gebrochenen Teil und wird deshalb nach Ausführung der INT-Funktion genauso wiedergegeben.

Die INT-Funktion wird häufig zum Runden von Zahlenwerten eingesetzt. Aus den oben genannten Gründen können aber nur positive Zahlen gerundet werden, während bei negativen ein Vorzeichenwechsel vorgenommen werden muß.

Ist die letzte nicht mehr auszugebene Ziffer 5 oder größer, findet eine Aufrundung, sonst eine Abrundung statt. Wir wollen jetzt ein kleines Programm schreiben, das gebrochene Dezimalzahlen auf ganze Zahlen rundet:

```
10 REM RUNDEN
20 INPUT "ZAHL"; Z
30 F=1 : IF Z < 0 THEN F= -1
40 Z = ABS(Z)
50 PRINT F * INT(Z+.5)
60 GOTO 20
```

In Zeile 20 wird die zu rundende Zahl eingegeben. Zeile 30 ordnet der Variablen F den Wert 1 zu. In der nachfolgenden IF-Anweisung wird geprüft, ob die Zahl negativ ist. In diesem Fall erhält die Variable F den Wert -1. Wir verwenden hier also F als Vorzeichenfaktor, der angibt, ob wir es mit einer positiven oder negativen Zahl zu tun haben und treffen die Vereinbarung, daß F bei positiven Zahlen den Wert 1 und bei negativen Zahlen den Wert -1 annehmen soll. Dies hat seinen guten Grund, denn wir führen in Zeile 40 die ABS-Funktion aus, die ein negatives Vorzeichen entfernt.

Die eigentliche Rundung findet in Zeile 50 mit Hilfe der INT-Funktion statt. Dazu wird zu Z der Wert .5 addiert und aus der Summe die nächstkleinere ganze Zahl gebildet. Betrachten wir einmal die Zahl 1.5 als Grenzfall des Auf- oder Abrundens. 1.5 + .5 ergibt 2, wovon die nächstkleinere ganze Zahl ebenfalls 2 ist. Es findet also eine Aufrundung statt. Dagegen wird 1.49 abgerundet, weil 1.49 + .5 = 1.99 ist, wovon die nächskleinere Zahl gleich 1 ist. Multiplizieren wir das Ergebnis mit dem Vorzeichenfaktor F, erhalten wir wieder das richtige Vorzeichen. Zeile 60 springt wieder an den Programmanfang nach Zeile 20, um die nächste Zahl anzufordern.

Hier nun ein Probelauf nach Eingabe von RUN:

```
ZAHL? 45.3
 45
ZAHL? -9.9
-10
ZAHL? -3.5
-4
ZAHL? 3.5
 4

usw.
```

Jetzt wollen wir unser Programm so umändern, daß es nicht auf ganze Zahlen, sondern auf zwei Stellen nach dem Dezimalpunkt rundet:

```
10 REM RUNDEN AUF ZWEI DEZIMALSTELLEN
20 INPUT "ZAHL"; Z
30 F=1 : IF Z < 0 THEN F= -1
40 Z = ABS(Z)
50 PRINT F * INT(Z*100+.5)/100
60 GOTO 20
```

Wir haben lediglich Zeile 50 abgewandelt. Da wir auf zwei Stellen runden, multiplizieren wir die Zahl zunächst mit 100, führen dann die INT-Funktion durch und teilen das Ergebnis wieder durch 100:

```
ZAHL? 123.456
 123.46
ZAHL? 12.999
 13
ZAHL? -45.673
-45.67
```

usw.

4.1.3 SGN

SGN ist eine Funktion, die das Vorzeichen ihres Arguments bestimmt oder angibt, ob es Null ist. Ist das Argument positiv, erzeugt SGN den Wert 1, ist es negativ, den Wert -1, und bei Null entsteht das Ergebnis 0. Hier einige Beispiele:

```
PRINT SGN(5.23)
 1

READY.

PRINT SGN(-78)
-1

READY

PRINT SGN(0)
0

READY.
```

Als wir im vorigen Absatz mit Hilfe der INT-Funktion Zahlen rundeten, mußten wir zunächst das Vorzeichen bestimmen, das wir in der Variablen F ablegten. Bei positiver Zahl war F = 1 und bei negativer -1.

In diesem Beispiel könnten wir Zeile 30 auch durch eine SGN-Funktion ersetzen, wobei die IF-Anweisung entfällt:

```
30 F = SGN(Z)
```

Der einzige Unterschied im Programmablauf mit der neuen Zeile 30 besteht darin, daß bei einem Zahlenwert Null F auch Null und nicht mehr 1 wird. Da aus der INT-Funktion in diesem Fall ebenfalls das Ergebnis Null hervorgeht, spielt es keine Rolle, ob diese Null nochmals mit einer anderen Null multipliziert wird.

4.1.4 SQR

Die SQR-Funktion erzeugt die Quadratwurzel ihres Arguments. Wird das Ergebnis mit sich selbst multipliziert (quadriert), entsteht wieder der Wert des Arguments:

```
PRINT SQR(49)
 7

READY.

PRINT SQR(2)
 1.41421356

READY.

PRINT SQR(-3)
?ILLEGAL QUANTITY ERROR
READY.
```

Im letzten Fall entstand ein ILLEGAL QUANTITY ERROR, der angibt, daß ein unzulässiger Zahlenwert vorliegt. Aus dem Argument -3 kann nämlich keine direkte Quadratwurzel gezogen werden.

Als Anwendungsbeispiel für die SQR-Funktion nachfolgend ein Programm, das die Diagonale eines Rechtecks berechnet. Hierzu müssen Länge und Breite quadriert, die Quadrate addiert und aus der Summe die Quadratwurzel gezogen werden (Satz von Pythagoras).

```
10 REM DIAGONALE IN RECHTECK
20 INPUT "LAENGE"; A
30 INPUT "BREITE"; B
40 C = SQR(A^2 + B^2)
50 PRINT "DIE DIAGONALE BETRAEGT"; C
```

Hier ein Probelauf:

```
LAENGE? 5
BREITE? 2
DIE DIAGONALE BETRAEGT 5.38516481

READY.
```

Beachten Sie bitte, daß es von der Rechenzeit her günstiger ist, die beiden Exponentiationen in Zeile 40 durch Multiplikationen zu ersetzen:

```
40 C = SQR(A*A + B*B)
```

In unserem Beispiel dürfte sich der Zeitvorteil durch Multiplikation noch weniger bemerkbar machen; anders ist es jedoch, wenn Sie riesige Datenmengen nach dieser Formel berechnen.

4.2 Wissenschaftliche Funktionen

Der Vollständigkeit halber wollen wir hier die wissenschaftlichen Funktionen nochmals betrachten, obwohl sie in Kapitel 2 bereits vorgestellt wurden. Falls Sie diese Funktionen nicht benötigen, können Sie den Abschnitt übergehen. Sie sind jedoch für alle Leser wichtig, die mit ihrem 128 PC naturwissenschaftlich-technische Berechnungen durchführen möchten.

BASIC bietet insgesamt sechs vorprogrammierte wissenschaftliche Funktionen, von denen X jeweils das Argument darstellt:

SIN(X) Sinus
COS(X) Cosinus
TAN(X) Tangens
ATN(X) Arcustangens
EXP(X) Potenz zur Zahl e (2.71...)
LOG(X) natürlicher Logarithmus

Beachten Sie, daß die Winkel für sämtliche trigonometrischen Funktionen im Bogenmaß angegeben werden müssen. Ein Beispiel zur Umrechnung befindet sich in Kapitel 2.

Die folgende Tabelle enthält zwar nicht alle wissenschaftlichen Funktionen, zumindest aber die wichtigsten, aus denen alle anderen leicht hergeleitet werden können. Hier einige Beispiele:

Funktion	Programmierung in BASIC
SECANS(X)	1/COS(X)
COSECANS(X)	1/SIN(X)
COTANGENS(X)	1/TAN(X)
ARCUSSINUS(X)	ATN(X/SQR(1-X*X))
ARCUSCOSINUS(X)	-ATN(X/SQR(1-X*X)) + PI/2
ARCUSCOTANGENS(X)	ATN(X) + PI/2
ARCUSSECANS(X)	ATN(X/SQR(X*X-1))
ARCUSCOSECANS(X)	ATN(X/SQR(X*X-1)) +(SGN(X)-1)*PI/2
SINUS HYPERBOLICUS(X)	(EXP(X)-EXP(-X))/2
COSINUS HYPERBOLIKUS(X)	(EXP(X)+EXP(-X))/2
TANGENS HYPERBOLIKUS(X)	EXP(-X)/(EXP(X)+EXP(-X))*2+1
COTANGENS HYP.(X)	EXP(-X)/(EXP(X)-EXP(-X))*2+1
SECANS HYP.(X)	2/(EXP(X)+EXP(-X))
COSECANS HYP.(X)	2/(EXP(X)-EXP(-X))
ARCUSSINUS HYP.(X)	LOG(X+SQR(X*X+1))
ARCUSCOSINUS HYP.(X)	LOG(X+SQR(X*X-1))
ARCUSTANGENS HYP.(X)	LOG((1+X)/(1-X))/2
ARCUSCOTANGENS HYP.(X)	LOG((X+1)/(X-1))/2
ARCUSSECANS HYP.(X)	LOG((SQR(1-X*X)+1)/X)
ARCUSCOSECANS HYP.(X)	LOG((SQR(1+X*X)+1)/X)
ZEHNERLOGARITHMUS(X)	LOG(X)/LOG(10)

Anmerkung: Die Zahl PI entspricht 3.14159265.

4.3 Zufallszahlen

Über die Funktion RND(X), die Zufallszahlen generiert, werden sich bestimmt diejenigen Leser freuen, die selbst Spiele programmieren möchten. Jedes Spiel wäre nämlich langweilig, wenn sein Ablauf vorhersehbar wäre und der Spieler würde bald die Lust verlieren.

Diese Zufallszahlen sind in Wirklichkeit nur Pseudo-Zufallszahlen, denn der Computer kann keine "zufällige" Entscheidung treffen. Sie sind durchaus errechnet und werden durch einen internen Taktzähler bestimmt, der u.a. auch die interene Uhr steuert.

Die von BASIC erzeugten Zufallszahlen liegen alle im Bereich zwischen 0 und .999999999. Wenn Sie die Anweisung

```
PRINT RND(X)
```

wiederholt ausführen, erhalten Sie jedesmal eine andere Zahl in diesem Bereich. Da es keine negativen Zufallszahlen gibt, läßt sich leicht mit Hilfe der INT-Funktion ein Würfelspiel simulieren, das ganzzahlige Zufallswerte zwischen 1 und 6 erzeugt:

```
10 GETKEY A$
20 Z = RND(X)
30 PRINT INT(6*Z)+1
40 GOTO 10
```

Ist dieses Programm gestartet, hängt es in einer Schleife. Zunächst wartet die GETKEY-Anweisung darauf, daß Sie eine beliebige Taste drücken. Zeile 20 ordnet dann der Variablen Z eine Zufallszahl zwischen 0 und .999999999 zu. Indem dieser Zahlenbereich mit 6 multipliziert wird, entstehen Zufallszahlen zwischen 0 und 5.99999999. Die INT-Funktion eliminiert nun den gebrochenen Teil, so daß ganze Zahlen zwischen 0 und 5 übrigbleiben. Zum Schluß wird noch 1 hinzuaddiert, so daß wir die gewünschten Zahlen zwischen 1 und 6 erhalten. Schließlich springt Zeile 40 wieder an den Programmanfang zurück.

Durch geringe Abänderung des Programms können wir Zufallszahlen in jedem beliebigen Bereich erzeugen. Wünschen wir z.B. Roulettzahlen zwischen 1 und 37, so lautet Zeile 30

```
30 PRINT INT(37*Z)+1
```

Kennen Sie sich erst einmal in der Programmierung von Zufallszahlen aus, sind Ihrer Phantasie keine Grenzen gesetzt, eigene Spiele zu schreiben. Zur Abrundung dieser Beschreibung hier noch ein Zahlenratespiel, bei dem eine Zahl zwischen 1 und 1000 zu erraten ist. Bei jedem Durchlauf bestimmt der Computer eine Zufallszahl in diesem Bereich. Daraufhin werden Sie aufgefordert, diese Zahl zu erraten und einzugeben.

Durch Hinweise erfahren Sie dann, ob diese Zahl, falls sie nicht stimmte, zu groß oder zu klein war und können dann solange einen weiteren Versuch starten, bis Sie die richtige Zahl erraten haben.

```
10 REM ZAHLENRATESPIEL
20 R = INT(1000*RND(X))+1
30 INPUT "BITTE ZAHL 1 - 1000 EINGEBEN"; Z
40 IF Z=R THEN PRINT "RICHTIG! HERZLICHEN
GLUECKWUNSCH!":END
50 IF Z>R THEN PRINT "ZU GROSS!":ELSE PRINT "ZU KLEIN!"
60 GOTO 30
```

Zeile 20 erzeugt zunächst eine Zufallszahl zwischen 1 und 1000. In Zeile 30 werden Sie aufgefordert, eine Zahl in diesem Bereich einzugeben. Haben Sie richtig geraten, erscheint die Meldung "RICHTIG! HERZLICHEN GLUECKWUNSCH!" und das Programm endet. Anderenfalls entsteht die Meldung "ZU GROSS!" bzw. "ZU KLEIN" und Sie werden aufgefordert, eine weitere Zahl einzugeben.

4.4 Frei definierte Funktionen

Neben den fest eingebauten Funktionen bietet BASIC die Möglichkeit, Funktionen selbst zu definieren und bei Bedarf aufzurufen und auszuführen. Dies geschieht mit der DEFFN-Anweisung, die vor dem ersten Aufruf definiert weden muß. Aus diesem Grunde stehen DEFFN-Anweisungen meistens am Programmanfang.

Eine Funktion frei zu definieren ist besonders dann sinnvoll, wenn sie häufig im Programm aufgerufen wird. Durch die Definition spart man nicht nur an Programmieraufwand, sondern auch an Speicherplatz. Kommt die hinter DEFFN stehende Funktion jedoch nur ein- oder zweimal vor, ist es meist sinnvoller, sie an der betreffenden Programmstelle als normale Anweisung auszuführen.

Einen Nachteil hat die DEFFN-Anweisung trotzdem, denn Sie darf nur eine Variable verwalten, die die Aufrufanweisung FN als Argument enthalten kann. Sollen z.B. in einem Programm häufig Zahlen gerundet werden, ist meist nur eine Variable nötig, die die zu rundende Zahl enthält und als Argument hinter FN stehen kann. Anders ist es bei der Volumenberechnung eines Quaders, in der Länge, Breite und Höhe in Variablen abgelegt sind. Zwar können wir auch diese Berechnung mit Hilfe einer Funk-

tionsdefinition durchführen, müssen aber die Variablenwerte gesondert eingeben. Wie dies funktioniert, zeigt das folgende Beispiel:

```
10 DEFFN V(X) = L * B * H
20 INPUT "LAENGE"; L
30 INPUT "BREITE"; B
40 INPUT "HOEHE"; H
50 PRINT "DAS VOLUMEN BETRAEGT"; FN V(X)
60 GOTO 20
```

In Zeile 10 wird die Funktion V definiert. V ist hier kein Variablen-, sondern der Funktionsname. Das X in Klammern ist in diesem Fall ein Blindargument, da wir die drei Variablen L, B und H gesondert eingeben.

Die Definition lautet nun, daß die Variablen L, B und H miteinander zu multiplizieren sind, wenn die Funktion in Form von FN V(X) aufgerufen wird. Dies geschieht in Zeile 50, nachdem den Variablen mit Hilfe von INPUT-Anweisungen Werte zugeordnet wurden.

Je nach Größe der Funktion benötigt die FN-Anweisung weit weniger Platz als die komplette Funktion.

Im folgenden Beispiel benötigen wir zur Quadratwurzelberechnung nur eine Variable X, die wir als Argument von DEFFN und FN verwenden wollen. Wichtig ist nur, daß sie in der Definition der Funktion enthalten ist:

```
10 DEFFN V(X) = SQR(X)
20 X = 25
30 PRINT FNV(X)
40 PRINT FNV(49)
50 A = 64
60 PRINT FNV(A)
```

In diesem Fall ist das Argument X auch Bestandteil der SQR-Funktion. Wird jetzt der Variablen X während des Programmablaufs ein Wert zugeordnet, wie hier in Zeile 20, dient dieser Wert bei Aufruf der Funktion als Argument (Zeile 30). Dabei kann das Argument auch aus einer Konstanten (Zeile 40) oder einer anderen Variablen bestehen (Zeile 60). Nach Ablauf des Programms erscheinen folgende Ergebnisse:

```
5
7
8

READY.
```

Abschließend noch ein Beispiel, das mit Hilfe einer Funktionsanweisung Zahlen auf zwei Stellen hinter dem Dezimalpunkt rundet:

```
10 DEFFN R(X) = INT(X*SGN(X)*100+.5)/100*SGN(X)
20 INPUT X
30 PRINT FNR(X)
40 GOTO 20
```

In diesem Programm wurde die gesamte Rundung geschickt in einer einzigen Funktion zusammengefaßt, wobei auf die ABS-Anweisung verzichtet wurde. Stattdessen wird die zu rundende Zahl mit ihrem eigenen Vorzeichen SGN(X) multipliziert, was in jedem Fall eine positive Zahl für die INT-Funktion ergibt. Abschließend wird das richtige Vorzeichen durch nochmalige Multiplikation mit SGN(X) wieder eingesetzt.

4.5 Unterprogramme

Frei definierte Funktionen können immer nur eine Programmzeile umfassen. Dies ist in vielen Fällen sinnvoll und völlig ausreichend. Nicht selten kommt es aber vor, daß ganze Programmteile, die aus einer Vielzahl von Zeilen bestehen, während des Programmablaufs wiederholt ausgeführt werden müssen.

Solche Programmteile können nun als Unterprogramm definiert und an jeder beliebigen Stelle des Hauptprogramms aufgerufen werden. Kein Wunder also, daß Unterprogramme noch mehr als Funktionen ein zeit- und platzsparendes Programmieren unterstützen.

Unterprogramme sind also komplett in sich abgeschlossene BASIC-Programme, die alle Befehle und Anweisungen enthalten dürfen. Sie müssen allerdings mit einer RETURN-Anweisung abschließen, um nach ihrer Abarbeitung wieder in das Hauptprogramm zurückzuspringen. Sinnvollerweise setzt man sie meist an das Programmende. Das Hauptprogramm muß dann allerdings mit END abschließen, um nicht in das Unterprogramm hineinzulaufen.

Jeder Aufruf erfolgt mit einer GOSUB-Anweisung, welcher die Zeilennummer folgt, mit der das Unterprogramm beginnt. Nach dem Rücksprung wird mit der Anweisung fortgefahren, die der GOSUB-Anweisung folgt. Hier ein einfaches Beispiel:

```
10 REM HAUPTPROGRAMM
20 GOSUB 100
30 GOSUB 200
40 GOSUB 300
50 GOSUB 200
60 END
100 REM ERSTES UNTERPROGRAMM
110 PRINT "HIER IST DAS ERSTE UNTERPROGRAMM"
120 RETURN
200 REM ZWEITES UNTERPROGRAMM
210 PRINT "HIER IST DAS ZWEITE UNTERPROGRAMM"
220 RETURN
300 REM DRITTES UNTERPROGRAMM
310 PRINT "HIER IST DAS DRITTE UNTERPROGRAMM"
320 RETURN
```

Nach Eingabe von RUN folgt

```
HIER IST DAS ERSTE UNTERPROGRAMM
HIER IST DAS ZWEITE UNTERPROGRAMM
HIER IST DAS DRITTE UNTERPROGRAMM
HIER IST DAS ZWEITE UNTERPROGRAMM

READY.
```

Zeile 20 ruft das erste Unterprogramm auf, das mit Zeile 100 beginnt. Nach dessen Abarbeitung fährt das Programm mit Zeile 30 fort, wo das zweite Unterprogramm aufgerufen wird. Das gleiche wiederholt sich in Zeile 40 mit dem dritten und in Zeile 50 nochmals mit dem zweiten Unterprogramm. Damit das Programm anschließend abbricht, steht in Zeile 60 die END-Anweisung. Würde sie fehlen, würde das Hauptprogramm ins erste Unterprogramm geraten. Da es in diesem Fall nicht durch GOSUB aufgerufen wurde, erscheint dann die Fehlermeldung

```
?RETURN WITHOUT GOSUB ERROR IN 120
READY.
```

4.6 Menütechnik

Vielleicht haben Sie schon einmal mit professionellen Programmen gearbeitet, die Sie fertig gekauft haben. Solche Programme zeichnen sich oft durch eine besonders gute Bedienerfreundlichkeit aus. Sie starten das Programm, und auf dem Bildschirm erscheinen Anweisungen, die Ihnen mitteilen, wie Sie das Programm nutzen können.

Diese Anweisungen werden in der Computertechnik auch Menü genannt. Sie sind so abgefaßt, daß auch derjenige damit arbeiten kann, der sonst über keine weiteren Programmierkenntnisse verfügt. Durch Drücken bestimmter Tasten werden einzelne Menüpunkte aufgerufen, die wiederum in verschiedene Teilprogramme oder untergeordnete Menüs verzweigen. Nach Abarbeitung der jeweiligen Teilroutinen gelangt man meistens wieder in das Hauptmenü zurück.

Auch wir wollen hier ein menügesteuertes Programm betrachten, das Volumenberechnungen für verschiedene Körper durchführt. Hier zunächst einmal das Programmlisting:

```
10 REM DIVERSE VOLUMENBERECHNUNGEN
20 PI = 3.14159265
30 PRINT"<CLR>"
40 PRINT"            VOLUMENBERECHNUNG"
50 PRINT
60 PRINT"QUADER                   1"
70 PRINT"ZYLINDER                 2"
80 PRINT"KEGEL                    3"
90 PRINT"PYRAMIDE                 4"
100 PRINT"KUGEL                   5"
110 PRINT
120 PRINT"E N D E                 E"
130 PRINT
140 PRINT"BITTE WAEHLEN SIE"
150 PRINT
160 GETKEY M$
170 IFM$="E" THEN END
180 M=VAL(M$)
190 IF M<1 OR M>5 THEN160
200 ON M GOSUB 250, 400, 540, 680, 820
210 GOTO 10
220 :
```

```
230 :
240 :
250 REM UNTERPROGRAMM FUER QUADER
260 PRINT "QUADER"
270 INPUT "LAENGE"; L
280 INPUT "BREITE"; B
290 INPUT "HOEHE"; H
300 PRINT
310 V=L*B*H
320 PRINT"DAS VOLUMEN BETRAEGT"; V
330 PRINT
340 PRINT"WEITER, BELIEBIGE TASTE DRUECKEN"
350 GETKEY A$
360 RETURN
370 :
380 :
390 :
400 REM UNTERPROGRAMM FUER ZYLINDER
410 PRINT"ZYLINDER"
420 INPUT "DURCHMESSER"; D
430 INPUT "HOEHE"; H
440 V = D*D*PI*H/4
450 PRINT
460 PRINT"DAS VOLUMEN BETRAEGT"; V
470 PRINT
480 PRINT"WEITER, BELIEBIGE TASTE DRUECKEN"
490 GETKEY A$
500 RETURN
510 :
520 :
530 :
540 REM UNTERPROGRAMM FUER KEGEL
550 PRINT"KEGEL"
560 INPUT "DURCHMESSER"; D
570 INPUT "HOEHE"; H
580 V = D*D*H*PI/12
590 PRINT
600 PRINT"DAS VOLUMEN BETRAEGT"; V
610 PRINT
620 PRINT"WEITER, BELIEBIGE TASTE DRUECKEN"
630 GETKEY A$
640 RETURN
```

```
650 :
660 :
670 :
680 REM UNTERPROGRAMM FUER PYRAMIDE
690 PRINT"PYRAMIDE"
700 INPUT"GRUNDSEITE"; L
710 INPUT"HOEHE"; H
720 V=L*L*H/3
730 PRINT
740 PRINT"DAS VOLUMEN BETRAEGT"; V
750 PRINT
760 PRINT"WEITER, BELIEBIGE TASTE DRUECKEN"
770 GETKEY A$
780 RETURN
790 :
800 :
810 :
820 REM UNTERPROGRAMM FUER KUGEL
830 PRINT"KUGEL"
840 INPUT "DURCHMESSER"; D
850 V=D*D*D/8*PI*4/3
860 PRINT
870 PRINT"DAS VOLUMEN BETRAEGT"; V
880 PRINT
890 PRINT"WEITER, BELIEBIGE TASTE DRUECKEN"
900 GETKEY A$
910 RETURN
```

Nachdem Sie dieses Programm gestartet haben, erscheint zunächst das Menü auf dem Bildschirm:

```
           VOLUMENBERECHNUNG

QUADER                        1
ZYLINDER                      2
KEGEL                         3
PYRAMIDE                      4
KUGEL5

E N D E                       E

BITTE WAEHLEN SIE
```

Sie haben nun die Möglichkeit, das Volumen wahlweise für Quader, Zylinder, Kegel, Pyramide oder Kugel zu berechnen. Diese Angaben, zusammen mit der dazugehörigen Ziffer, nennt man Menüpunkte. Drücken Sie nun eine 1, gelangen Sie in das Teilprogramm, welches das Volumen eines Quaders berechnet. Wenn Sie die 2 drücken, können Sie das Volumen eines Zylinders berechnen usw. Möchten Sie das Programm abbrechen, drücken Sie die Taste E.

Uns interessiert hier in erster Linie der Aufbau dieses menügesteuerten Programms:

Zunächst geben die Zeilen 10 bis 150 das Menü auf dem Bildschirm aus. Zeile 20 ordnet der Variablen PI den Wert 3.14159265 zu, die wir für Kreisberechnungen benötigen. Der 128 PC besitzt zwar auch eine Pi-Taste, die diesen Wert erzeugt, jedoch wollen wir es der besseren Übersichtlichkeit halber hier so belassen. Vor der Ausgabe des Menüs löscht Zeile 30 den Bildschirm.

Zeile 160 wartet jetzt mit GETKEY M$ auf einen Tastendruck. Dabei wird der Stringvariablen M$ ein Wert zugeordnet. Die IF-Anweisung in Zeile 170 prüft nun, ob Sie ein "E" eingegeben haben, wenn Sie das Programm verlassen möchten. In diesem Fall wird dann die Anweisung END ausgeführt.

Die eigentlichen Menüpunkte 1 bis 5 werden zunächst ebenfalls in der Stringvariablen M$ gespeichert, weil die GETKEY-Anweisung nur Strings annehmen kann. Für uns ist es aber wesentlich bequemer, wenn wir die Zahlenwerte in Fließkommaschreibweise zur Verfügung haben. Die Umformung nimmt Zeile

```
180 M = VAL(M$)
```

vor. Enthält das Argument der VAL-Anweisung Ziffernwerte, die auch für numerische Werte zulässig sind, formt sie diese ins numerische Fließkommaformat um, so daß wir mit ihnen mathematische Operationen ausführen können. In unserem Fall liegen nun also die eingegebenen Ziffern im Fließkommaformat vor und sind der Variablen M zugeordnet.

Jedes gute Menü führt auch eine Prüfung durch, ob die eingegebenen Werte zulässig sind. In unserem Programm übernimmt dies Zeile 190:

```
190 IF M<1 OR M>5 THEN 160
```

Die IF-Anweisung fragt, ob der Variablenwert M kleiner als 1 oder größer als 5 ist, denn wir können hier ja nur die Menüpunkte 1 bis 5 verarbeiten. Ist eine dieser beiden Bedingungen erfüllt (Boolsche Operation OR), springt das Programm wieder zurück in Zeile 160 zur GETKEY-Anweisung und erwartet einen neuen Tastendruck. Dieser Vorgang ist von außen her nicht sichtbar. Drücken Sie nämlich eine falsche Taste, geschieht scheinbar gar nichts, denn das Programm hängt in einer Warteschleife.

Sämtliche Teilprogramme, die die einzelnen Volumenberechnungen durchführen, sind hier als Unterprogramme definiert und schließen deshalb jeweils mit einer RETURN-Anweisung ab. Ihr eigentlicher Aufbau braucht uns an dieser Stelle nicht weiter zu interessieren, denn solche Programme haben wir in ähnlicher Form schon kennengelernt. Der besseren Übersicht halber haben wir die einzelnen Programmsegmente durch bedeutungslose Leerzeichen unterteilt, die nur aus der Zeilennummer und einem Doppelpunkt bestehen.

Was uns aber interessiert, ist die Art und Weise, wie die Teilprogramme aufgerufen werden. Wir wissen bereits, daß die GOSUB-Anweisung ein Unterprogramm aufruft und dieses nach der Ausführung wieder an die Aufrufstelle zurückkehrt. In Zeile 200 finden wir GOSUB in einer anderen Form vor:

```
200 ON M GOSUB 250, 400, 540, 680, 820
```

Nun wird uns auch klar, warum wir die Zahlenwerte ins numerische Format umgewandelt haben. Diese Anweisung fragt nämlich den Wert der Fließkommavariablen M ab und entscheidet dann aufgrund dieses Wertes, welches Unterprogramm aufgerufen wird:

Wert von M	Unterprogramm ab Zeile
1	250
2	400
3	540
4	680
5	820

Somit entspricht also die erste Zahl hinter ON M GOSUB der Zeile, die aufgerufen wird, wenn M den Wert 1 erhält, die zweite, wenn M den Wert 2 erhält usw. Dasselbe hätten wir auch mit folgenden 5 IF-Anweisungen erreicht:

```
IF M = 1 THEN GOSUB 250
IF M = 2 THEN GOSUB 400
IF M = 3 THEN GOSUB 540
IF M = 4 THEN GOSUB 680
IF M = 5 THEN GOSUB 820
```

Wir sehen aber, daß die ON...GOSUB-Anweisung bequemer zu programmieren ist und weniger Speicherplatz beansprucht.

Nach Abarbeitung der einzelnen Unterprogramme springt die GOTO-Anweisung in Zeile 210 wieder an den Programmanfang und gibt das Menü erneut aus.

In diesem Zusammenhang sei noch erwähnt, daß es für GOTO die ähnliche Anweisung ON...GOTO gibt. Hier ein Beispiel:

```
100 ON A GOTO 100, 200, 300
```

Ist der Inhalt der Variablen A gleich 1, findet ein Sprung in Zeile 100 statt, ist er gleich 2, wird in Zeile 200 gesprungen usw.

4.7 Fehlersuche und Fehlerbehandlung

In diesem Abschnitt wollen wir uns mit der Fehlersuche und der Fehlerbehandlung beschäftigen. Jeder Programmierer, selbst der erfahrenste, bringt kaum ein Programm zustande, das auf Anhieb fehlerfrei läuft. Dabei nimmt die Wahrscheinlichkeit, daß Fehler auftreten, bei längeren und komplizierteren Programmen zu und es wird immer schwieriger, die Fehler zu lokalisieren.

Viele Fehler entstehen aus Unachtsamkeit bei der Programmierung. So wird z.B. ein Variablenname oder Schlüsselwort falsch geschrieben oder eine Zahl falsch eingegeben. Wie beim legendären zerstreuten Professor hat schon mancher A gesagt, B geschrieben, C gedacht und in Wirklichkeit war D gemeint. Manch einer war bereits nahe daran, seine Nerven zu verlieren, wenn ein Programm nicht so ablief, wie er es wollte.

Wir können aber lernen, auf die richtige Weise mit den Fehlern umzugehen oder besser noch, sie von vornherein zu vermeiden. So gibt es verschiedene Arten von Fehlern, die wir im folgenden kurz betrachten wollen.

Da sind einmal die Leichtsinnsfehler, die durch mangelnde Konzentration oder Unachtsamkeit entstehen. Subtilere und deshalb gewöhnlich schwerwiegendere Fehler sind solche, die auf mangelndem Verständnis für die Arbeitsweise von bestimmten Anweisungen beruhen, die dem Computer erteilt werden. Oft wird verkannt, daß eine bestimmte Situation auftreten kann, die es gilt, in den Griff zu bekommen. Irren ist menschlich und ein Programmierer ist ein Mensch, der Computer dagegen nicht.

Als weiteres gibt es solche Fehler, die nicht auf menschlichem Irrtum beruhen und bei der Programmerstellung bewußt mit eingeplant werden. Komfortablere Anwenderprogramme enthalten z.B. oft Anweisungen, die feststellen sollen, ob ein Peripheriegerät, wie Drucker oder Floppy, eingeschaltet ist. Ist dies nämlich nicht der Fall, gibt der Computer eine Fehlermeldung aus, die abgefragt wird, um eventuell geeignete Gegenmaßnahmen zu ergreifen. Bei nicht eingeschaltetem Drucker könnte dies so aussehen, daß der Bediener aufgefordert wird, das Gerät einzuschalten.

Derartige Maßnahmen werden derweilen noch drastischer praktiziert. Viele Softwareanbieter möchten nämlich nach Möglichkeit verhindern, daß ihre Programme auf Diskette unzulässigerweise vervielfältigt werden. Dazu bringen sie auf der Diskette extra Fehler an. Wird nun versucht, dieses Programm zu kopieren und "normal" zu starten, ist es nicht lauffähig, da bestimmte eingebaute "Fehler" nicht vorhanden sind, die aber zur einwandfreien Arbeitsweise des Programms unbedingt erforderlich sind. Im Extremfall kann in einer solchen Situation die Diskette sogar zerstört werden.

Eine Grundvoraussetzung zur Vermeidung von Fehlern ist eine klare Zielsetzung von dem, was ein Programm alles ausführen soll. Viele Programmierer verwenden zur Planung spezielle Fluß- oder Ablaufdiagramme, die jede einzelne Anweisung enthalten. Wir wollen hier wegen unserer relativ einfachen Beispiele auf derartige Diagramme verzichten, es aber dennoch nicht versäumen, ein Konzept aufzustellen. Betrachten wir hierzu nochmals das Programm, das für fünf verschiedene Körper das Volumen berechnet. Das Konzept dazu könnte folgendermaßen aussehen:

1. Zielsetzung: Was für ein Programm will ich schreiben? Zu welchem Zweck möchte ich es einsetzen?

   Antwort: Ich wünsche ein Programm, das das Volumen von Quadern, Zylindern, Kegeln, Pyramiden und Kugeln berechnet.

2. Aufbau: Aus welchen Teilelementen soll sich das Programm zusammensetzen?

   Antwort: Eingabemenü

   Unterprogramm für Quader
   Unterprogramm für Zylinder
   Unterprogramm für Kegel
   Unterprogramm für Pyramide
   Unterprogramm für Kugel

3. Detaillierte Ausarbeitung für jedes Teilelement:

Eingabemenü:	Bildschirm löschen Menüpunkte ausgeben Menüpunkte abfragen Gewähltes Unterprogramm ausführen Zurück ins Eingabemenü
Unterprogramm für Quader:	Meldung "Quader" ausgeben Eingabewerte lesen (Länge, Breite, Höhe) Volumen berechnen Volumen ausgeben Warteschleife auf Tastendruck Rückkehr ins Hauptprogramm

Auf die Ausarbeitung der anderen Unterprogramme wollen wir hier verzichten, da sie ähnlich aufgebaut sind.

Nun können wir uns an den Computer setzen, geeignete Anweisungen wählen und schließlich das Programm eingeben. Bevor wir es ablaufen lassen, speichern wir es sicherheitshalber auf Diskette oder Kassette ab.

Wir haben das Programm zwar jetzt im Computer, aber mit hoher Wahrscheinlichkeit läuft es noch nicht fehlerfrei. Dank unseres strukturierten Aufbaus sind wir aber in der Lage festzustellen, in welchem Teilprogramm ein Fehler auftritt.

Häufig haben wir die Schreibregeln für die BASIC-Programmiersprache verletzt. In einem solchen Fall erscheint die Fehlermeldung

?SYNTAX ERROR IN (Zeilennummer)

READY.

und das Programm bricht ab. Nun können wir die fehlerhafte Zeile listen und korrigieren. Wird anschließend die HELP-Taste gedrückt, erscheint zusätzlich die falsche Anweisung in reverser Darstellung. Diese Taste kann im Anschluß an jede Fehlermeldung gedrückt werden.

Manchmal ist es sinnvoll, das Programm an bestimmten Stellen an-zuhalten, um den Inhalt einiger Variablen zu überprüfen. Gibt z.B. das Unterprogramm für Quader ein falsches Ergebnis aus, fügen wir folgende Zeile ein, die wir später wieder löschen:

```
315 STOP
```

Das Programm hält jetzt jedesmal in Zeile 315 an und meldet sich mit

```
BREAK IN 315
READY.
```

Nun können wir prüfen, ob die Variablen L, B, H und V die richtigen Werte oder Zwischenergebnisse enthalten. Dies geschieht ganz normal mit der PRINT-Anweisung im Direktmodus. Wir können uns aber Tipparbeit sparen, indem wir statt PRINT nur ein Fragezeichen setzen, z.B.

```
? V
5.6

READY.
```

Wurde ein Programm mit BREAK unterbrochen, kann es mit

```
CONT
```

fortgesetzt werden. In unserem Fall würde es dann mit Zeile 320 fortfahren. CONT funktioniert allerdings nicht, wenn bei der Unterbrechnung Änderungen im Programm vorgenommen wurden. Dann nämlich erscheint die Fehlermeldung

```
?CAN'T CONTINUE ERROR
READY.
```

4.7.1 Ablaufverfolgung

Eine zusätzliche Hilfe zur Auffindung von Fehlern ist die Ablaufverfolgung. Ist sie eingeschaltet, erscheinen beim Programmablauf die Nummern der gerade abgearbeiteten Zeilen auf dem Bildschirm. Die Ablaufverfolgung ist eine große Hilfe, wenn festgestellt werden soll, ob Programmverzweigungen richtig funktionieren. Jeder Fehler kann dann anhand der ausgegebenen Zeilennummern sofort festgestellt werden.

Die Ablaufverfolgung wird mit

```
TRON
```

eingeschaltet, worauf das Programm normal mit RUN zu starten ist.

```
TROFF
```

schaltet sie wieder ab. TRON und TROFF können aber auch in Programme eingebaut und somit im Programmodus betrieben werden.

4.7.2 Programmierte Fehlerbehandlung

Hier behandeln wir nun die Fehler, die bewußt mit eingeplant werden. BASIC 7.0 stellt uns Anweisungen zur Verfügung, mit der wir Fehler auffangen und bearbeiten können. Dies geschieht zunächst einmal mit der TRAP-Anweisung, der eine Zeilennummer folgt. Tritt nun ein Fehler auf, springt das Programm in diese Zeile und führt dann die Fehlerbehandlung durch. Anschließend kehrt RESUME wieder an die Stelle zurück, an der die Fehlermeldung auftrat.

Für den Einsatz der TRAP-Anweisung gibt es viele Anwendungsbereiche. So kann das Programm z.B. feststellen, ob während einer Berechnung Ergebnisse auftreten, die größer als der zulässige Wertebereich von ca. 1E+38 sind oder ob die Quadratwurzel aus einem negativen Wert gezogen werden soll.

Im Normalfall bricht das Programm bei Auftreten eines Fehlers ab und gibt eine entsprechende Fehlermeldung aus. Bei früheren BASIC-Versionen war es oft ärgerlich, wenn Programme mit langer Laufzeit aufgrund von Fehlern plötzlich abbrachen und nach deren Behebung neu gestartet werden mußten. Stellen Sie sich einmal vor, Sie arbeiten mit einem riesigen

Adressenverwaltungsprogramm, lesen Hunderte von Adressen von der Diskette ein, schreiben neue Adressen hinzu und wollen schließlich die gesamte Liste ausdrucken. Aber leider haben Sie vergessen, den Drucker einzuschalten, das Programm bricht ab und gibt die Fehlermeldung

?DEVICE NOT PRESENT ERROR IN (Zeilennummer)

aus. Ein nachträgliches Einschalten des Druckers nützt nichts, die ganze Arbeit war umsonst und Sie müssen von vorne beginnen.

Um Ihnen solche frustrierenden Enttäuschungen in Zukunft zu ersparen, haben wir hier ein kleines Programm, das abfragt, ob der Drucker eingeschaltet ist. Die darin enthaltenen Anweisungen können in jedes andere Programm übernommen werden.

```
10 REM ABFRAGE, OB DRUCKER EINGESCHALTET
20 TRAP100
30 OPEN4,4,0," "
40 PRINT "DRUCKER EINGESCHALTET"
50 CLOSE 4
60 END
70 :
80 :
90 :
100 REM FEHLERBEHANDLUNGSROUTINE
110 CLOSE4
120 PRINT ER;ERR$(ER);" ERROR";" IN ZEILE";EL
130 PRINT"BITTE DRUCKER EINSCHALTEN UND"
140 PRINT"BELIEBIGE TASTE DRUECKEN"
150 GETKEY A$
160 RESUME
```

Zeile 20 enthält die TRAP-Anweisung mit der Zeilennummer 100. Tritt nun ein Fehler auf, was in diesem Fall bedeutet, daß der Drucker nicht eingeschaltet ist, verzweigt das Programm in Zeile 100. Ist der Drucker dagegen eingeschaltet, findet keine Verzweigung statt und das Programm läuft normal weiter. In unserem Fall erscheint dann die Bestätigung "DRUCKER EINGESCHALTET" auf dem Bildschirm.

In der Fehlerbehandlungsroutine schließt Zeile 110 das noch geöffnete logische File. Zeile 120 enthält nun die reservierten Variablennamen ER, ERR$(ER) und EL. Dabei enthält ER die interne Nummer des Fehlers,

ERR$(ER) die Fehlermeldung im Klartext und EL die Nummer der Zeile, in der der Fehler auftrat. Diese Werte geben wir nun in übersichtlicher Form in einer PRINT-Anweisung aus, so daß im Falle des nicht eingeschalteten Druckers die Meldung

5 DEVICE NOT PRESENT ERROR IN ZEILE 30

erscheint. Dabei gibt die 5 am Zeilenanfang an, daß wir es hier mit dem Fehler Nr. 5 zu tun haben (DEVICE NOT PREDENT).

Statt dieser Meldung hätten wir auch eine beliebige andere oder gar keine Meldung ausgeben können. Die im Normalfall, also ohne TRAP-Anweisung entstehende Fehlermeldung, die einen Programmabbruch zur Folge hat, erscheint hier also nicht automatisch.

Am Schluß der Fehlerbehandlungsroutine werden Sie aufgefordert, den Drucker einzuschalten und dann eine beliebige Taste zu drücken (GETKEY-Anweisung). Die Routine schließt mit

160 RESUME

ab, was bedeutet, daß die Steuerung an die Stelle zurückgegeben wird, an der der Fehler auftrat. Dabei wird die gleiche Anweisung, die den Fehler verursachte, nochmals ausgeführt. In unserem Fall ist es die OPEN-Anweisung, die den Ausgabekanal zum Drucker öffnet.

Von RESUME gibt es noch zwei weitere Varianten. Wenn es nicht wünschenswert ist, die fehlerverursachende Anweisung nochmals auszuführen, geben Sie

RESUME NEXT

an. Dabei fährt das Programm erst mit der nächstfolgenden Anweisung fort. Darüberhinaus besteht aber auch die Möglichkeit, das Programm im Anschluß an die Fehlerbehandlung an einer beliebigen anderen Stelle fortzusetzen. Zu diesem Zweck ist hinter RESUME die Nummer der Zeile anzugeben, in die gesprungen werden soll, z.B.:

RESUME 1000

Abschließend noch ein wichtiger Hinweis, der nicht in allen Handbüchern und Beschreibungen enthalten ist. Denjenigen Lesern, die bereits über Pro-

grammierkenntnisse verfügen, ist vielleicht aufgefallen, daß wir in unserem Beispiel die sehr ungewöhnliche Anweisung OPEN 4,4,0," " zum Öffnen des Druckerkanals benutzt haben. Die sonst übliche und meist völlig ausreichende Anweisung OPEN 4,4 erzeugt bei ausgeschaltetem Drucker keine Fehlermeldung und führt auch zu keinem Programmabbruch. Einerseits mag dies erfreulich erscheinen, andererseits aber ist für diese OPEN-Anweisung auch keine Fehlerbehandlung möglich und unser Programm gibt bei ausgeschaltetem Drucker die Meldung "DRUCKER EINGESCHALTET" aus. Achten Sie bitte bei derartigen Abfragen darauf, daß eine Sekundäradresse und ein Filename mit mindestens einem Zeichen in der OPEN-Anweisung angegeben ist.

5 Listenverarbeitung

In den vergangenen Kapiteln haben wir nur mit relativ kleinen Datenmengen gearbeitet, die wir über die INPUT-Anweisung oder über DATA-Zeilen ins Programm eingelesen haben. Zu Übungszwecken mag dies für Sie vielleicht ganz interessant gewesen sein, zum Aufbau einer professionellen Dateiverwaltung gehört jedoch noch einiges mehr.

Programme, die große Datenmengen verarbeiten, nennt man Datenverarbeitungs- oder Datenerfassungsprogramme. Sie arbeiten meist mit einem Massenspeicher, der in vielen Fällen aus einem oder mehreren Floppylaufwerken, oft aber auch aus einer Festplatte besteht, die weitaus mehr Daten als eine Diskette aufnehmen kann.

Um ein "echtes" Datenerfassungsprogramm zu schreiben, reicht aber unser bisheriges Wissen noch nicht aus; dazu benötigen wir weitere Kenntnisse und Erfahrungen, die in diesem Kapitel behandelt werden. Den Massenspeicher wollen wir vorläufig allerdings nicht berücksichtigen, sondern uns statt dessen zur Eingabe von Daten weiterhin noch mit DATA-Zeilen begnügen.

Auf den folgenden Seiten finden sie alles, was Sie zu einer Listen- oder Tabellenverarbeitung benötigen. Zunächst lernen wir verschiedene Stringverarbeitungstechniken und die formatierte Ausgabe kennen. Anschließend befassen wir uns mit Feldern, die eine Vielzahl von Datenelementen enthalten können und die einen wichtigen Platz in der Datenverarbeitung einnehmen. Nicht zu kurz kommt dabei auch das Auffinden und Sortieren von Daten, die in Feldern abgelegt sind, wodurch die Datenverarbeitung erst richtig interessant wird.

5.1 Stringmanipulationen

In den vergangenen Kapiteln haben wir bereits wiederholt mit Strings gearbeitet. Wir wissen, daß sie einen gesonderten Datentyp darstellen und daß Stringvariablen Zeichenketten enthalten, die in der Regel aus ASCII-Zeichen bestehen.

Allerdings ist uns noch wenig über Anweisungen bekannt, die Stringmanipulationen durchführen, d.h. Strings aneinanderreihen, trennen oder in sonstiger Weise beeinflussen.

Gegenüber der reinen Zahlenverarbeitung bringt die Stringverarbeitung eine gewisse Problematik mit sich. Wir kennen bereits die drei Datentypen Fließkomma, Integer und String. Fließkomma- und Intergerzahlen benötigen immer die gleiche Menge an Speicherplatz. Ändert sich ihr Variablenwert, so wird der alte Wert einfach von dem neuen überschrieben. Aus diesem Grund können numerische Variablen während des Programmablaufs immer an der gleichen Stelle im Speicher plaziert sein.

Bei Strings ist dies anders, da sie verschiedene Längen annehmen können. Ein String im Commodore BASIC kann beispielsweise zwischen 0 und 255 Zeichen enthalten.

Dies wäre unerheblich, wenn am Anfang des Programms die jeweiligen Stringvariablen eine feste Länge erhielten, die dann, wie bei den numerischen Variablen, während des gesamten Programmablaufs konstant bleibt.

Wie wir jedoch gleich sehen werden, können die Längen der Stringvariablenwerte sich ständig verändern. Betrachten wir dazu ein einfaches Beispiel:

```
10 INPUT "BITTE EIN NETTES WORT EINGEBEN"; A$
20 PRINT A$
30 PRINT "DANKESCHOEN!"
40 GOTO 10
```

Dieses Programm hängt in einer Schleife und fordert Sie jedesmal erneut auf, ein nettes Wort einzugeben. Dies geschieht mit der INPUT-Anweisung in Zeile 10, wo der eingegebene String der Variablen A$ zugeordnet wird. Lassen wir jetzt das Programm einmal ablaufen:

```
BITTE EIN NETTES WORT EINGEBEN? HI!
HI!
DANKESCHOEN
BITTE EIN NETTES WORT EINGEBEN? GUTEN MORGEN!
GUTEN MORGEN!
DANKESCHOEN
BITTE EIN NETTES WORT EINGEBEN? HALLO - WIE GEHT'S?
HALLO - WIE GEHT'S?
DANKESCHOEN
usw.
```

Im ersten Durchgang wurde das Wort "HI!" eingegeben, das insgesamt drei Zeichen umfaßt, im zweiten das Wort "GUTEN MORGEN!" mit 13 Zeichen und schließlich "HALLO - WIE GEHT'S?" mit 19 Zeichen. Die Länge des Variableninhalts von A$ ändert sich also ständig.

Um dieses Problem in den Griff zu bekommen, haben sich die Erfinder des BASIC-Interpreters etwas einfallen lassen. Im Speicher befindet sich eine sogenannte Variablentabelle, in der alle Variablen, gleich welchen Typs, eingetragen sind. Da numerische Variablen immer den gleichen Speicherplatz benötigen, sind ihre Werte gleich mit in der Variablentabelle enthalten. Für Stringvariablen ist in der Tabelle lediglich ein Zeiger enthalten, der die Länge und die Anfangsadresse des Strings im Speicher enthält. Die Werte von Stringvariablen füllen den noch freien Speicher von oben nach unten, d.h. von der höchsten zur niedrigsten nicht anderweitig belegten freien Adresse, auf.

Wird nun eine neue Stringvariable angelegt, so füllt ihr Wert immer mehr den Speicher nach unten hin auf. Das gleiche geschieht auch mit der Variablen A$ in unserem Programm, wenn sie einen neuen Wert erhält. Dabei wird der Zeiger, auch Descriptor genannt, auf den neuen Wert gesetzt. Der alte Wert von A$ wird also nicht überschrieben und bleibt als "Stringmüll" zurück.

Es ist daher einleuchtend, daß Programme, die viele Strings erzeugen oder umändern, den Speicherplatz immer mehr nach unten hin auffüllen und gleichzeitig eine Menge "Stringmüll" nicht mehr benötigter Strings zurücklassen. Irgendwann ist dann der Zeitpunkt gekommen, daß es so nicht mehr weitergehen kann.

Diejenigen Leser, die bereits mit dem Commodore 64 gearbeitet haben, haben vielleicht schon einmal bemerkt, daß der C-64 mitten in einem Programm plötzlich für einige Sekunden seine Arbeit unterbricht. Dies geschieht besonders bei solchen Programmen, die viel mit Strings arbeiten.

In dieser Pause führt der Computer eine Stringmüllbeseitigung (garbage collect) durch. Dabei werden alle Strings, die noch gültig, d.h. in der Variablentabelle aufgeführt sind, nach oben hin neu geordnet und aneinandergefügt, damit wieder genügend Platz für neue Strings frei wird.

Je nach Anzahl der sich im Speicher befindlichen Strings ist die Stringmüllbeseitigung eine Frage der Zeit. Während sie beim C-64 noch relativ lange dauerte, braucht sie beim 128 PC höchstens eine Sekunde. Der 128 PC verwaltet nämlich die Strings auf eine andere Weise als der C-64, um dem Zeitproblem bei der Stringmüllbeseitigung Rechnung zu tragen.

Eine Stringmüllbeseitigung wird umso seltener auftreten, je größer der BASIC-Speicher ist und umso weniger Platz das Programm einschließlich der Variablentabelle benötigt. Beim 128 PC ist der BASIC-Speicher darüber hinaus noch in zwei feste Hauptteile unterteilt, nämlich in den Programm- und in den Variablenspeicher. Der Variablenspeicher umfaßt ca. 64KB und ist um einiges größer als der Gesamt-BASIC-Speicher des C-64.

Während man bei früheren Commodore-Rechnern mit der FRE-Anweisung die Stringmüllbeseitigung künstlich hervorrufen und den noch freien Speicherplatz bestimmen konnte, ist dies beim 128 PC nur bedingt möglich. Wenn Sie hier

```
?FRE(0)
```

eingeben, sagt Ihnen der Computer, wieviele Bytes im Programmspeicher noch frei sind. Er gibt jedoch keine Auskunft über den noch freien Platz im Variablenspeicher. Sie sehen dies besonders deutlich, wenn Sie bereits gleich nach dem Einschalten den noch freien Speicherplatz abfragen:

```
?FRE(0)
58109

READY.
```

Die Zahl 58109, die Sie hier erhalten, entspricht exakt der Größe des Programmspeichers, während Ihnen die Einschaltmeldung sagt, daß insgesamt 122365 Bytes für Programm- und Variablenspeicher zur Verfügung stehen.

5.1.1 Stringverkettung

Das Pluszeichen dient in BASIC nicht nur zur Addition von Zahlenwerten oder als Vorzeichen, sondern auch zur Verkettung von Strings. Somit ist es möglich, aus Einzelstrings einen Gesamtstring zu erzeugen, der allerdings nicht mehr als 255 Zeichen enthalten darf. Hier ein Beispiel:

```
10 A$ = "COMMODORE"
20 B$ = "128"
30 C$ = "PERSONAL"
40 D$ = "COMPUTER"
50 G$ = A$ + B$ + C$ + D$
60 PRINT G$
```

Nach Eingabe von RUN erscheint:

```
COMMODORE128PERSONALCOMPUTER

READY.
```

Hier wurde aus den vier Einzelstrings A$, B$, C$ und D$ der Gesamtstring G$ gebildet. In der vorliegenden Form sollten wir ihn aber nicht belassen und deshalb Leerzeichen zwischen die einzelnen Wörter setzen. Aus diesem Grunde schreiben wir Zeile 50 etwas um:

```
50 G$ = A$ + " " + B$ + " " + C$ + " " + D$
```

Jetzt erscheint G$ in übersichtlicher Form:

```
COMMODORE 128 PERSONAL COMPUTER

READY.
```

5.1.2 LEN

Mit LEN können Sie die Länge eines Strings bestimmen. Als Beispiel wollen wir nochmals unser letztes Programm betrachten. Nachdem Sie es ausgeführt haben, also sämtliche Variablen angelegt sind, geben Sie folgendes ein (statt PRINT setzen wir hier ein Fragezeichen):

```
? LEN(G$)
 31

READY.

? LEN(A$)
 9

READY.
```

Wir haben soeben die Längen der Strings G$ und A$ bestimmt. Der Gesamtstring G$ ist 31 Zeichen, der String A$ 9 Zeichen lang.

Die LEN-Anweisung sollten Sie immer dann verwenden, wenn es darum geht, die Länge von Strings zu prüfen. Dies kann besonders bei Programmen mit vielen Stringoperationen der Fall sein.

Beim Versuch, einen String mit mehr als 255 Zeichen Länge zu erzeugen, entsteht die Fehlermeldung

```
?STRING TOO LONG ERROR IN (Zeilennummer)
```

5.1.3 LEFT$ und RIGHT$

Betrachten wir folgendes Beispiel:

```
A$ = "ABCDEFGHIJKLMN"

READY.
```

```
? LEFT$(A$,5)
ABCDE

READY.

? RIGHT$(A$,3)
LMN

READY.
```

Damit erklären sich die Anweisungen LEFT$ und RIGHT$ eigentlich schon von selbst. So erzeugt LEFT$(A$,X) einen neuen String, der aus den ersten X Zeichen des Strings A$ besteht. Bei RIGHT$(A$,X) entsteht ebenfalls ein neuer String, der aber die rechten X Zeichen des Strings A$ enthält.

5.1.4 MID$

MID$(A$,X,Y) erzeugt einen String aus A$, der bei der Position X beginnt und Y Zeichen lang ist. Auch hierzu ein Beispiel:

```
A$ = "ABCDEFGHIJKLMN"

READY.

? MID$(A$,8,7)
HIJKLMN

READY.
```

Hier wird aus dem String A$ ein neuer String gebildet, der mit dem 8. Zeichen beginnt und 7 Zeichen lang ist.

MID$ kann aber auch in umgekehrter Weise Verwendung finden, indem wir in einen gegebenen einen neuen String einsetzen. Hier ein Beispiel.

```
A$ = "********************"

READY.
```

```
MID$(A$,8,6) = "HALLO!"
READY.

? A$
*******HALLO!*******

READY.
```

In diesem Fall wird der String "HALLO!" in den String A$ eingesetzt und erscheint dort ab dem 8. Zeichen.

Bei allen Stringoperationen ist es egal, ob die Strings als Konstante oder Variable vorliegen. Im letzen Beispiel hätten wir auch den String "HALLO!" der Variablen C$ zuordnen und den Inhalt von A$ ausschreiben können:

```
MID$("********************",8,6) = C$
```

5.1.5 INSTR

Mit dieser Anweisung ist es möglich, einen String in einem anderen String zu suchen, und dessen Position anzugeben, z.B.:

```
A$ = "MARKT & TECHNIK - VERLAG"

READY.

B$ = "TECHNIK"

READY.

? INSTR(A$,B$)
 9

READY.
```

Hier wird untersucht, ob der String B$ im String A$ enthalten ist. Dies ist tatsächlich der Fall, denn das Wort "TECHNIK" beginnt ab Position 9 des Strings A$. Ist der gesuchte String dagegen nicht enthalten, erscheint das Ergebnis 0.

5.1.6 STR$ und VAL

STR$ wandelt einen numerischen Wert in einen String um:

```
A = 1000

READY.

S$ = STR$(A)

READY.

? S$
 1000

READY.

? LEN(S$)
 5

READY.
```

Hier haben wir den numerischen Wert 1000, der in der Variablen A abgelegt ist, in einen String mit dem Namen S$ umgewandelt. Nach außen hin macht es keinen Unterschied, ob wir die Zahl 1000 als String oder als numerische Variable auf dem Bildschirm ausgeben.

Mit Hilfe von LEN stellen wir fest, daß S$ 5 Zeichen lang ist, obwohl die Zahl 1000 nur 4 Ziffern besitzt. Dies hat durchaus seine Richtigkeit, denn die STR$-Anweisung reserviert immer eine Stelle für das Vorzeichen. Ist das Vorzeichen negativ, erscheint vorne ein Minuszeichen, ist es dagegen positiv, erscheint ein Leerzeichen (Blank).

VAL ist die Umkehrfunktion von STR$ und formt einen als String eingegebenen Zahlenwert ins Fließkommaformat um. Hierzu ein Beispiel:

```
10 INPUT "1.ZAHL"; A$
20 INPUT "2.ZAHL"; B$
30 A = VAL(A$)
40 B = VAL(B$)
50 PRINT A * B
```

Ein Probelauf liefert

```
1. ZAHL? 5
2. ZAHL? 4
 20

READY.
```

In diesem Programm haben wir die beiden Operanden in Form von Strings eingegeben. Um mit ihnen mathematische Operationen ausführen zu können, müssen sie zunächst einmal ins Fließkommaformat umgewandelt werden. Dies geschieht in Zeile 30 und 40, so daß Zeile 50 schließlich die Multiplikation ausführen kann.

5.1.7 CHR$ und ASC

Sämtlichen Zeichen, einschließlich den unsichtbaren Steuerzeichen, ist intern ein äquivalenter Zifferncode zugeordnet, den man auch ASCII-Code nennt. Wir wollen jetzt einmal versuchen, diesen Code dem Computer zu entlocken:

```
? ASC ("A")
 65

READY.

? ASC ("B")
 66

READY.

? ASC ("Z")
 90

READY.
```

Wir haben dazu die ASC-Anweisung benutzt, die zu jedem Buchstaben den entsprechenden ASCII-Code erzeugt. Ohne diese Prozedur nun für alle 26 Buchstaben durchführen zu müssen, können wir uns leicht vorstellen, daß der Buchstabe C den Code 67, D den Code 68 und Y den Code 89 besitzt.

Diese Codes wollen wir nun dazu verwenden, das Alphabet in etwas ungewöhnlicher Form auf dem Bildchirm auszugeben:

```
10 REM AUSGABE DES ALPHABETS
20 FOR I = 65 TO 90
30 PRINT CHR$(I);
40 NEXT
```

Nach RUN erscheint:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ

READY.
```

Dieses Programm enthält die CHR$-Anweisung, die aus den ASCII-Codes die entsprechenden lesbaren Zeichen erzeugt.

Wir wissen bereits, daß Anführungszeichen einen String abgrenzen, aber daß sie sonst nie in ihm enthalten sein dürfen. Es gibt jedoch einen Trick, dies doch zu ermöglichen, denn das Anführungszeichen entspricht dem ASCII-Code 34:

```
10 A$ = "SIE SAGTE "+CHR$(34)+"GUTEN MORGEN!"+CHR$(34)
20 PRINT A$
```

Nach der Ausführung erscheint

```
SIE SAGTE "GUTEN MORGEN!"

READY.
```

Sie sehen also, daß Sie auch "unzulässige" Zeichen mit CHR$ erzeugen können.

Abschließend noch ein String, der Sie verblüffen wird.

```
10 A$ = "ERSTE ZEILE"+CHR$(13)+"ZWEITE ZEILE"
20 PRINT A$
```

Nach der Ausführung erscheint:

```
ERSTE ZEILE
ZWEITE ZEILE

READY.
```

Dieser String wird also in zwei Zeilen untereinander ausgegeben. Dies liegt an dem ASCII-CODE 13, der einen Wagenrücklauf und Zeilenvorschub veranlaßt (Carriage Return). Wenn Sie diesen Code in Form von CHR$(13) ausgeben, geschieht dasselbe wie beim Drücken der RETURN-Taste.

5.1.8 Interne Uhr

Als letzte Stringanweisung wollen wir hier TI$ behandeln. TI$ ist ein reservierter Variablenname und dient zur Abfrage der internen Uhr. Wenn wir die interne Uhr stellen, müssen wir TI$ einen sechsstelligen Ziffernwert zuordnen. Dabei stehen die beiden ersten Ziffern für die Stunden, die beiden mittleren für die Minuten und die beiden rechten für die Sekunden. Nachfolgend nun ein Programm, das eine Digitaluhr simuliert:

```
10 REM DIGITALUHR
20 INPUT "BITTE UHRZEIT EINGEBEN"; TI$
30 PRINT "<CLR>"
40 T1$ = LEFT$(TI$,2)
50 T2$ = MID$(TI$,3,2)
60 T3$ = RIGHT$(TI$,2)
70 PRINT "ES IST JETZT";
80 PRINT T1$;" UHR ";T2$;" MIN. ";T3$;" SEK.";
90 PRINT "<1 x CRSR UP>"
100 GOTO 40
```

In Zeile 20 geben Sie die augenblickliche Uhrzeit ein, um die Uhr zu stellen. Dabei müssen Sie unbedingt 6 Ziffern eingeben, da sonst ein ?ILLEGAL QUANTITY ERROR auftritt. Hier einige Beispiele zu TI$:

Uhrzeit			Eingabe
1 Uhr	3 Min.	6 Sek.	010306
12 Uhr	0 Min.	0 Sek.	120000
15 Uhr	17 Min.	56 Sek.	151756

Ist TI$ einmal richtig eingestellt, kann zu jedem Zeitpunkt die Uhrzeit abgefragt werden, wozu auch unser Programm dient. Damit die Uhrzeit in übersichtlicher Form erscheint, werden mit Hilfe der Stringoperationen LEFT$, MID$ und RIGHT$ die Strings T1$, T2$ und T3$ für Stunden, Minuten und Sekunden gebildet und mit einem entsprechenden Vermerk in Zeile 70 und 80 ausgegeben. Zeile 90 setzt den Cursor um eine Zeile nach oben, so daß der Anschein ensteht, es handele sich um eine feste Anzeige, die nicht den Bildschirm hinaufläuft. Zeile 100 springt wieder in Zeile 40, wo derselbe Vorgang von neuem beginnt.

5.1.9 Laufschrift

Mit Hilfe der verschiedenen Stringoperationen können wir eine Laufschrift auf dem Bildschirm erzeugen. Hier ein Beispiel:

```
10 REM LAUFSCHRIFT
20 A$=" NEUE COMPUTERBUECHER SIND GANZ TOLL! "
30 PRINT "<CLR, 12 x CRSR DOWN>"
40 A$=A$+A$
50 I=I+1 : IF I=41 THEN I=1
60 PRINT MID$(A$,I,40)
70 PRINT "<2 x CRSR UP>"
80 FOR J=1 TO 100:NEXT
90 GOTO 50
```

Das Programm funktioniert denkbar einfach. Der als Laufschrift auszugebende String steht in Zeile 20 und hat eine Länge von 40 Zeichen. Er ist also genauso lang wie eine Bildschirmzeile im 40-Zeichen-Modus. Zeile 30 setzt den Cursor um 12 Zeilen nach unten, so daß er etwa in der Bildschirmmitte steht.

Zeile 40 reiht den auszugebenden String zweimal aneinander, so daß er jetzt eine Gesamtlänge von 80 Zeichen hat. Der ganze Trick besteht darin, 40 Zeichen des Strings in einer Schleife wiederholt auszugeben. Bei jedem Durchgang verschiebt sich das erste auszugebende Zeichen im String um eine Stelle nach rechts, was durch die MID$-Anweisung in Zeile 60 gesteuert wird. Wird anschließend der Cursor um eine Zeile nach oben gesetzt, entsteht durch die kontinuierliche Ausgabe der Eindruck einer Laufschrift.

5.2 Formatierte Ausgabe

Die bisher betrachteten PRINT-Anweisungen konnten zwar Werte auf dem Bildsschirm ausgeben, hinterließen aber nicht immer das beste Erscheinungsbild. Zwar konnten wir das Aussehen etwas verbessern, indem wir hinter PRINT ein Komma oder Semikolon setzten, die Übersichtlichkeit war deshalb aber noch lange nicht gegeben.

Gerade wer Listen und Tabellen mit dem Computer ausgeben möchte, weiß es zu schätzen, wenn in Zahlenkolonnen die Dezimalpunkte untereinander stehen. Mit den bisher behandelten Rundungsverfahren und Stringoperationen könnten wir sicher eine solche Formatierung erreichen, wenn auch auf etwas umständliche Weise. Viel einfacher geht es aber mit der PRINT USING-Anweisung, mit der wir uns gleich näher befassen werden.

Da umfangreiche Listen und Tabellen selten nur auf dem Bildschirm ausgegeben werden, wollen wir uns in diesem Zusammenhang auch mit der Druckerausgabe beschäftigen. Dabei spielt es keine Rolle, welchen Drucker Sie benutzen. Die meisten Drucker sind zwar für die formatierte Listenausgabe ausgerüstet, benötigen jedoch dazu diverse Steuerzeichen, die sich von Fabrikat zu Fabrikat unterscheiden. Die hier vorgestellten Formatierungsanweisungen sind aber so aufgestellt, daß sie gleichermaßen für Bildschirm und Drucker Gültigkeit haben.

Zuvor noch ein paar Worte zur Druckeransteuerung, über die es aber nicht allzuviel zu sagen gibt. Im Gegensatz zur Bildschirmausgabe sieht sie folgendermaßen aus:

1. Logisches File öffnen
2. Daten mit PRINT# ausgeben
3. Logisches File schließen.

Hier ein Beispiel, das die Zahlen 1 bis 10 auf dem Drucker ausgibt:

```
10 OPEN 1,4
20 FOR I = 1 TO 10
30 PRINT#1, I;
40 NEXT
50 CLOSE 1
```

Zeile 10 öffnet das logische File mit der Nummer 1 auf den Drucker mit der Geräteadresse 4. Die Zeilen 20 bis 40 geben die Zahlen aus, wobei statt PRINT die Anweisung PRINT#1, Verwendung findet. Dieses PRINT# mit Rautenzeichen bezieht sich immer auf ein logisches File, das angegeben werden muß. In unserem Fall ist es das logische File 1, das wir in Zeile 10 mit OPEN 1,4 geöffnet haben. Außerdem muß hinter PRINT#1 ein Komma stehen. Nachdem alle Werte ausgegeben sind, muß das logische File 1 mit CLOSE 1 wieder geschlossen werden. Wenn Sie diese Regeln beachten, können Sie alles, was Sie mit PRINT auf dem Bildschirm schreiben, auch auf Ihrem Drucker ausgeben. Dies gilt auch für die Ausgabe mit PRINT USING, bei der Sie PRINT#1, USING für die Druckerausgabe schreiben. Vergessen Sie aber nicht, das logische File zu öffnen und zu schließen!

5.2.1 PRINT USING

Mit PRINT USING können wir Zahlen und Strings formatiert ausgeben. Dabei sind die auszugebenden Stellen in einem Formatstring mit Rautenzeichen oder Nummernzeichen (#) vorzugeben. Hier einige Beispiele:

```
PRINT USING "###.##"; 3.187
  3.19

READY.

A = -3.14159 : PRINT USING "####.####"; A
  -3.1420

READY.
```

Manchmal ist es aber notwendig, ein Vorzeichen mit auszugeben. Es muß dann im Formatstring angegeben sein und kann wahlweise vorne oder hinten stehen:

```
PRINT USING "+###.##"; 6.23
+  6.23

READY.

PRINT USING "###.##-"; -17.3
 17.30-

READY.
```

Mit PRINT USING können auch Zahlenwerte in Exponentialschreibweise ausgegeben werden. Zu diesem Zweck setzt man hinter das Rautenzeichen vier Pfeile:

```
PRINT USING "+#.##^^^^"; 78.98
+7.90E+01

READY.
```

Darüber hinaus besteht für Strings noch die Möglichkeit, sie entweder zentriert oder rechtsbündig auszugeben:

```
PRINT USING "########="; "ABC"
'   ABC   '

READY.

PRINT USING "########>"; "ABC"

'      ABC'

READY.
```

Die Apostrophe erscheinen normalerweise nicht und dienen hier nur zur Kennzeichnung der Ausgabe. Ist also das letzte Zeichen des Formatstrings ein Gleichheitszeichen, wird der String zentriert, bei einem ">"-Zeichen wird er rechtsbündig ausgegeben.

Ist ein auszugebender String länger als im Formatstring vorgesehen, werden die überhängenden Zeichen rechts abgeschnitten:

```
PRINT USING "##########"; "ARTIKEL-NUMMER"
ARTIKEL-NU

READY.
```

Abschließend wollen wir noch ein kleines Programm betrachten, das eine Preisliste mit Artikelnummer, Artikelbezeichnung und Preis ausgeben soll. Die Artikelnummer wird hierin als String behandelt:

```
10 REM PREISLISTE
20 PRINT"NR.    ARTIKEL    PREIS"
30 PRINT
40 DO
50 READ N$
60 IFN$ = "9999" THEN EXIT
70 READ AT$, P
80 PRINT USING"####";N$;
90 PRINT"  ";
100 PRINT USING"##########";AT$;
110 PRINT"  ";
120 PRINT USING"####.##";P
130 LOOP
140 DATA 1581,TISCH,125
150 DATA 2659,STUHL,89.5
160 DATA 0435,SESSEL,239.5
170 DATA 5437,REGAL,134.95
180 DATA 3436,LAMPE,43.7
190 DATA 9999
```

Nach Eingabe von RUN erscheint die Preisliste folgendermaßen auf dem Bildschirm:

```
NR.     ARTIKEL          PREIS

1581    TISCH           125.00
2659    STUHL            89.50
0435    SESSEL          239.50
5437    REGAL           134.95
3436    LAMPE            43.70

READY.
```

Der Programmaufbau ist uns aus früheren ähnlichen Beispielen schon bekannt. Neu sind hier lediglich die PRINT USING-Anweisungen zwischen Zeile 80 und 120. Indem sie jeweils mit einem Semikolon abschließen, lassen sich mehrere formatierte Ausgabewerte in eine Reihe setzen. Diese Möglichkeit kennen wir bereits von der normalen PRINT-Anweisung her.

Die einzelnen Daten sind wieder in DATA-Zeilen vorgegeben, wobei der letzte Wert 9999 die Abbruchbedingung in Zeile 60 darstellt.

Wollen Sie umfangreiche Listen ausdrucken, empfiehlt es sich, vorher einen Ausdruckplan anzufertigen. Dieser ist ein gitterartiges Schema, dessen Reihen den Zeilen und dessen Spalten den Druckpositionen in der Zeile entsprechen. Die verschiedenen Titel, Spaltenüberschriften und Einzelposten werden in den Ausdruckplan genauso eingetragen, wie sie später auf dem Bildschirm oder auf dem Papier erscheinen. Dabei ist es sinnvoll, die einzelnen Positionen durch ihren entsprechenden Formatstring auf dem Ausdruckplan zu kennzeichnen.

Für Ausdruckpläne verwenden Sie spezielle Formulare, die es im Fachhandel zu kaufen gibt oder aber einfaches, kariertes Papier. Achten Sie auch darauf, daß Sie pro Zeile nicht mehr Positionen angeben, als Ihr Drucker ausgeben kann. Die meisten Drucker können 80 Zeichen, einige Sonderausführungen 132 oder mehr Zeichen pro Zeile ausdrucken, was unter Umständen auch von der eingestellten Schriftbreite abhängt.

5.3 Windows

Der Commodore 128 PC bietet die Möglichkeit, auf dem Bildschirm Textfenster (Windows) einzurichten. Ist dies einmal geschehen, finden sämtliche Bildschirmoperationen in diesem Textfenster statt, das wie ein eigener Bildschirm zu betrachten ist.

Auf dem Bildschirm können mehrere Windows mit unterschiedlicher Größe erscheinen. Viele professionelle Programme wenden die Window-Technik an, um im oberen oder unteren Teil des Bildschirms verschiedene Anweisungen zu geben, die der Bediener ständig vor Augen hat, solange er mit dem Programm arbeitet.

Auf ein einmal definiertes Fenster beziehen sich alle Anweisungen, die in irgendeiner Weise Einfluß auf den Bildschirm haben, wie z.B. INPUT, PRINT oder GETKEY. Solche Anweisungen, zu denen auch das Löschen des Bildschirms oder die Cursorbewegungen gehören, beeinflussen nur das Window selbst, nicht aber den restlichen Bildschirmbereich.

Textfenster werden mit der WINDOW-Anweisung erzeugt, die folgendermaßen definiert werden muß:

```
WINDOW Spalte oben links, Zeile oben links,
       Spalte unten rechts, Zeile unten rechts,
       <Löschen des Windows>
```

Das Löschen des Windows ist wahlweise und geschieht nur, wenn hier eine 1 angegeben ist. Ansonsten dürfen die Spalten- und Zeilenangaben sich nur innerhalb der jeweiligen Bildschirmgrenze bewegen. Der 40-Zeichen-Bildschirm umfaßt die Zeilen 0 bis 24 und die Spalten 0 bis 39. Beim 80-Zeichen-Bildschirm liegt der Zeilenbereich ebenfalls zwischen 0 und 24, der Spaltenbereich dagegen zwischen 0 und 79.

Als Anwendungsbeispiel wollen wir ein Textfenster erzeugen, das ringsherum einen Rand von vier Zeichen auf dem Bildschirm frei läßt. Im Fenster selbst geben wir in einer Endlosschleife das Wort "TEXTFENSTER" aus, auf dem Rand erscheint das Wort "RAND":

```
10 PRINT "<CLR>"
20 PRINT
30 PRINT"                  RAND"
40 WINDOW 4, 4, 35, 20
50 PRINT"TEXTFENSTER ";: GOTO 50
```

Zeile 10 löscht den noch vollständigen Bildschirm, auf den in Zeile 30 das Wort "RAND" geschrieben wird. Das Fenster wird in Zeile 40 definiert, worauf Zeile 50 als Endlosschleife das Wort "TEXTFENSTER" im Fenster selbst immer wieder ausgibt. Falls Sie im 80-Zeichen-Modus arbeiten, ändern Sie Zeile 40 folgendermaßen ab:

```
40 WINDOW 4, 4, 75, 20
```

Nachdem Sie die Wirkungsweise der WINDOW-Anweisung kennengelernt haben, brechen Sie das Programm mit der STOP-Taste ab. Wir werden jetzt die RWINDOW-Anweisung kennenlernen, die uns die Parameter unseres Fensters angibt:

```
PRINT RWINDOW(0)
 16

READY.

PRINT RWINDOW(1)
 31

READY.

PRINT RWINDOW(2)
 40

READY.
```

In der RWINDOW-Anweisung liefert das Argument 0 die Zeilenanzahl und das Argument 1 die Spaltenanzahl des Fensters. Bei einem Argument von 2 erscheint die Zahl 40 oder 80, je nachdem, ob wir mit dem 40- oder 80-Zeichen-Bildschirm arbeiten.

In unserem Fall hat das Fenster also 16+1 Zeilen und 31+1 Spalten; die Bildschirmbreite umfaßt 40 Zeichen. Bei den Zeilen und Spalten müssen wir jeweils eins hinzuzählen, da die Zählung hier bei Null beginnt. Somit hat ein Zeichen in der linken oberen Ecke die Position Spalte 0, Zeile 0.

Wir können dieses Fenster wieder verlassen, indem wir zweimal hintereinander die HOME-Taste drücken. Beim ersten Drücken bezieht sich HOME auf das Textfenster, d.h., der Cursor befindet sich in dessen linker oberer Ecke. Drücken wir HOME ein zweites Mal, steht er in der linken oberen Bildschirmecke und sämtliche Anweisungen beziehen sich wieder auf den gesamten Bildschirm.

5.4 Felder

Allen Variablen, die wir bisher betrachtet haben, konnte immer nur ein Wert gleichzeitig zugeordnet werden. Dies galt gleichermaßen für Fließkomma-, Integer- und Stringvariablen. Jetzt lernen wir Variablen kennen, die sich auf ganze Listen oder Wertebereiche gleichzeitig beziehen. Solche Variablen heißen indizierte Variablen oder auch Felder und können vom Datentyp Fließkomma, Integer oder String sein.

In Feldern lassen sich große Datenmengen ablegen und verarbeiten. Betrachten wir zunächst nochmals unser Beispiel, das die Zahlen 1 bis 10 auf dem Bildschirm ausgibt:

```
10 FOR I = 1 TO 10
20 PRINT I;
30 NEXT
```

Wenn wir dieses Programm ausführen, wird in jedem Schleifendurchgang der jeweilige Inhalt der Laufvariablen I ausgegeben, der sich beim nächsten Durchgang schon wieder ändert. I ist hier also variabel und behält nie einen fest zugeordneten Wert.

Es kann nun erforderlich sein, daß sämtliche Werte, die I bei der Abarbeitung dieser Schleife annimmt, ständig abrufbereit gespeichert sind. Dazu legen wir ein eindimensionales Feld mit einem einzigen Variablennamen an.

Diesem Feld wollen wir den Namen A geben und es soll zehn Kästchen oder Elemente besitzen, in denen wir unsere Werte ablegen. Das erste Kästchen erhält somit die Zahl 1, das zweite die Zahl 2 usw. und das zehnte die Zahl 10.

Feldvariablennamen sind prinzipiell genauso aufgebaut wie diejenigen "normaler Variablen". Sie unterscheiden sich aber durch einen angefügten Index, der in Klammern stehen muß. Hier nun ein Beispiel, in dem wir die Zahlen 1 bis 10 der Feldvariablen A zuordnen:

```
10 DIM A(10)
20 A(1) = 1
30 A(2) = 2
40 A(3) = 3
50 A(4) = 4
60 A(5) = 5
70 A(6) = 6
80 A(7) = 7
90 A(8) = 8
100 A(9) = 9
110 A(10) = 10
```

Feldvariablen werden nicht automatisch angelegt, sondern benötigen eine besondere Anweisung, die ihnen genügend Speicherplatz reserviert. Dies geschieht in unserem Beispiel in Zeile 10 mit der DIM-Anweisung. Mit DIM A(10) wird BASIC aufgefordert, eine Feldvariable A mit Platz für zehn Elemente anzulegen. Diesen Vorgang nennt man Dimensionierung.

Die restlichen Zeilen ordnen den verschiedenen Elementen der Feldvariablen die entsprechenden Werte zu. Nach Ausführung des Programms machen wir im Direktmodus einige Stichproben, um festzustellen, ob die Werte auch richtig zugeordnet wurden:

```
PRINT A(3)
 3

READY.

PRINT A(8)
 8

READY.

PRINT A(1)
 1

READY.
```

Wir sehen also, daß die Elemente die richtigen ihnen zugeordneten Werte enthalten, die wir jederzeit natürlich auch im Programmodus abrufen können. Normalerweise brauchen wir den Elementen aber nicht durch Einzelanweisungen Werte zuordnen, sondern können dies auch in einer Schleife tun.

Das folgende Programm enthält zwei Schleifen, von denen die erste den Feldelementen Werte zuordnet, die von der zweiten wieder aufgerufen und ausgegeben werden.

```
10 DIM A (10)
20 FOR I = 1 TO 10
30 A(I) = I
40 NEXT
50 :
60 FOR I = 1 TO 10
70 PRINT A(I);
80 NEXT
```

Nach Eingabe von RUN erfolgt

```
1 2 3 4 5 6 7 8 9 10

READY.
```

Wir können Zeile 30 auch abändern, daß jedes Element den fünffachen Wert seines Indices erhält:

```
30 A(I) = 5 * I
```

Wenn wir das Programm jetzt ausführen, erhalten wir:

```
5 10 15 20 25 30 35 40 45 50

READY.
```

Strenggenommen haben wir mit der DIM-Anweiung nicht Platz für 10, sondern für 11 Elemente geschaffen, denn bei jeder Dimensionierung entsteht immer auch ein Feld mit dem Index 0. In unserem Fall hat das Element A(0) jedoch keine Bedeutung.

Wenn Sie den Versuch unternehmen, mehr Elemente zu belegen, als durch die DIM-Anweisung vorgegeben sind, entsteht die Fehlermeldung

```
?BAD SUBSCRIPT ERROR IN (Zeilennummer)
```

Dimensionieren Sie also die Felder immer ausreichend groß, daß Sie alle Daten, die Sie benötigen, darin unterbringen können. Am besten führen Sie dies gleich am Programmanfang durch. Achten Sie aber darauf, daß jedes Feld nur einmal dimensioniert werden darf. Wird nämlich der Versuch unternommen, dies ein zweites Mal zu tun, erscheint die Fehlermeldung

?REDIM'D ARRAY ERROR IN (Zeilennummer)

und das Programm bricht ab. Da der verfügbare Speicher begrenzt ist, können Sie ein Feld auch nicht beliebig groß dimensionieren.

Nun wollen wir noch zwei praxisbezogene Beispiele betrachten. Zunächst nehmen wir uns dafür wieder das Programm vor, das eine Preisliste ausgibt. Wir ändern es aber so um, daß sämtliche Daten nicht nur ausgegeben, sondern darüberhinaus auch in Feldern abgelegt werden:

```
10 REM PREISLISTE
15 DIM N$(20), AT$(20), P(20)
17 I = 1
20 PRINT"NR.    ARTIKEL    PREIS"
30 PRINT
40 DO
50 READ N$(I)
60 IF I>20 OR N$(I) = "9999" THEN EXIT
70 READ AT$(I), P(I)
80 PRINT USING"####";N$(I);
90 PRINT"  ";
100 PRINT USING"##########"; AT$(I);
110 PRINT"  ";
120 PRINT USING"####.##";P(I)
125 I = I+1
130 LOOP
140 DATA 1581,TISCH,125
150 DATA 2659,STUHL,89.5
160 DATA 0435,SESSEL,239.5
170 DATA 5437,REGAL,134.95
180 DATA 3436,LAMPE,43.7
190 DATA 9999
```

Statt einfacher Variablen verwenden wir hier die Feldvariablen N$(I) für die Artikelnummer, AT$(I) für den Artikel und P(I) für den Preis. Zeile 15 enthält eine DIM-Anweisung, die alle drei Felder gleichzeitig dimensioniert. Für jedes Feld wurden hier absichtlich 20 Elemente vorgesehen, um die Preisliste noch erweitern zu können. Zeile 17 setzt den Zähler I auf 1, der vor der LOOP-Anweisung jeweils um eins erhöht wird.

Das nächste Programm wertet Temperaturmessungen aus und bestimmt den höchsten und den niedrigsten Temparaturwert.

```
10 REM TEMPERATURAUSWERTUNG
20 DIM T(20)
30 FOR I = 1 TO 20
40 READ T(I)
50 NEXT
60 L = T(1)
70 H = T(1)
80 FOR I = 2 TO 20
90 IF T(I) > H THEN H = T(I)
100 IF T(I) < L THEN L =T(I)
110 NEXT
120 PRINT "DIE TIEFSTE TEMPERATUR WAR"; L
130 PRINT "DIE HOECHSTE TEMPERATUR WAR"; H
140 DATA 12.4, 13.7, 10.3, 18.8, 15.0
150 DATA 14.2, 15.5, 17.4, 16.6, 14.9
170 DATA 11.4, 13.5, 12.8, 11.9, 10.2
180 DATA 13.6, 14.8, 17.1, 13.6, 15.0
```

Nach Eingabe von RUN erscheint:

```
DIE TIEFSTE TEMPARATUR WAR 10.2
DIE HOECHSTE TEMPERATUR WAR 18.8

READY.
```

Zunächst werden die verschiedenen Temparaturwerte eingelesen und im Feld T(I) abgelegt. Die niedrigste (L) und die höchste Temperatur (H) werden dann mit dem ersten Wert T(1) vorbelegt. In der FOR...NEXT-Schleife werden sämtliche Werte vom zweiten Wert an gelesen und geprüft, ob sie größer als der bisher höchste oder kleiner als der bisher tiefste Wert sind. Wenn ja, werden sie zum neuen höchsten bzw. tiefsten Wert bestimmt.

Felder besitzen die wichtige Eigenschaft des Direktzugriffs. Dies bedeutet, daß man einzelne Elemente nach Belieben lesen oder beschreiben kann, ohne die davorliegenden Elemente erst einlesen zu müssen. Wir können diese Eigenschaft bei allen hier aufgeführten Beispielen beobachten, die Datenwerte aus DATA-Zeilen einlesen. Ist der Einlesevorgang beendet, kann auf jedes Element frei zugegriffen werden.

Neben den eindimensionalen gibt es auch mehrdimensionale Felder. Sie sind durch mehrere durch Kommas getrennte Indices gekennzeichnet. Hier ein Beispiel, das ein zweidimensionales Feld dimensioniert:

```
10 DIM P(25,10)
20 FOR J = 1 TO 25
30 FOR I = 1 TO 10
40 P(J,I) = J*I
50 NEXT I
60 NEXT J
```

Zeile 10 dimensioniert das Feld P mit 25 mal 10 Elementen, denen in Zeile 40 Werte aus dem Produkt von I und J zugeordnet werden.

5.5 Suchen und Sortieren

Felder eignen sich besonders für Such- und Sortiervorgänge. Stellen Sie sich einmal vor, Sie suchen einen Namen aus einer unsortierten Adressenliste heraus. Dabei gehen Sie die Liste von oben nach unten durch und vergleichen jeden einzelnen Namen mit dem, den Sie suchen. Dies machen Sie solange, bis Sie entweder Ihren Namen gefunden haben oder die Liste zu Ende ist.

Auch der Computer kann ein Feld nach diesem Schema durchsuchen, was man als sequentielles (aufeinanderfolgendes) Suchen bezeichnet. Hier ein kleines Unterprogramm, das die sequentielle Suche durchführt und das man an jedes andere Programm anfügen kann. Dabei kennzeichnet N$(I) das zu durchsuchende Feld und NA$ den Namen, der gesucht werden soll:

```
1000 REM SEQUENTIELLES SUCHEN
1010 I = 1
1020 DO
1030 IF N$(I) = NA$ OR N$(I) = "ENDE" THEN EXIT
1040 I = I + 1
1050 LOOP
1060 RETURN
```

Bei diesem Programm wird die Suche solange fortgeführt, bis entweder der gesuchte Name gefunden wurde oder das Tabellenende erreicht ist. Dies erkennt das Programm an der Markierung "ENDE", die im ersten nicht mehr benutzten Feldelement vorhanden sein muß.

Darüber hinaus gibt es noch das binäre Suchen, das sich aber nur für sortierte Felder eignet. Es ist damit vergleichbar, wenn Sie in einem Lexikon ein Stichwort suchen. Zunächst schlagen Sie das Lexikon in der Mitte auf,

wodurch es in zwei Hälften geteilt wird. Nun wird das erste Wort auf der ersten Seite der zweiten Hälfte betrachtet. Erscheint Ihr Stichwort davor, so ist es in der ersten Hälfte zu suchen, anderenfalls befindet es sich in der zweiten Hälfte.

Diesen Vorgang wiederholen Sie nun für die Hälfte des Lexikons, in der das gesuchte Wort steht. Sie wird wieder in der Mitte aufgeschlagen und es wird erneut bestimmt, welche der beiden Hälften das Wort enthält. Dies wiederholen Sie so lange, bis nur noch eine Seite übrigbleibt.

Binäres Suchen in einem Lexikon mag eine etwas umständliche Methode darstellen, in Computerprogrammen ist es aber dem sequentiellen Suchen weit überlegen, besonders bei großen Dateien. So benötigt ein Verzeichnis mit 1000 Einträgen bei sequentieller Suche durchschnittlich 500, maximal jedoch 1000 Vergleiche, um einen Eintrag zu finden, während es bei der binären Suche nie mehr als 10 sind.

Auch für das binäre Suchen wollen wir ein kleines Unterprogramm betrachten:

```
1000 REM BINAERES SUCHEN
1010 L = 1 : MI = 1 : H = FR -1
1020 DO WHILE L<=H AND N$(MI)<>NA$
1030 MI = INT((L+H)/2)
1040 IF NA$ < N$(MI) THEN H = MI-1
1050 IF NA$ > N$(MI) THEN L = MI+1
1060 LOOP
1070 I=MI
1080 RETURN
```

Dabei ist wieder N$(I) das Feld mit den Namen und NA$ der gesuchte Name. L und H grenzen den Bereich der Elemente ein, in denen sich der gesuchte Name befindet. MI zeigt jeweils auf die Mitte des zu halbierenden Bereichs. I gibt die Nummer des Elements, welches den gesuchten Namen enthält, an das aufrufende Hauptprogramm zurück. FR ist das erste Element im Feld, das nicht benutzt wird.

Nun wollen wir noch zwei Sortierverfahren für Strings kennenlernen, die aber in leicht abgewandelter Form auch für numerische Werte einzusetzen sind. Bei Strings verwenden wir dazu ein Stringfeld und bei numerischen Werten ein Feld für Fließkommazahlen.

Zunächst befassen wir uns mit dem Sortieren durch Einfügen. Während des ganzen Sortiervorganges wird das Feld in zwei Teile unterteilt, nämlich in einen bereits sortierten und in einen unsortierten Teil. Aus dem unsortierten Teil wird immer das erste Element herausgenommen, mit den Elementen im sortierten Teil verglichen und dort an der richtigen Stelle eingefügt. Hier zunächst einmal das Programmlisting:

```
10 REM SORTIEREN DURCH EINFUEGEN
20 :
30 :
40 :
50 REM DATEN EINLESEN
60 DIM N$(20)
70 I=1
80 DO
90 READ N$(I)
100 IF N$(I)="ENDE" THEN K=I-1:EXIT
110 I=I+1
120 LOOP
130 :
140 :
150 :
160 REM DATEN UNSORTIERT AUSGEBEN
170 PRINT"NAMEN UNSORTIERT"
180 PRINT
190 FOR I=1 TO K
200 PRINT N$(I)
210 NEXT
220 PRINT
230 PRINT
240 :
250 :
260 :
270 REM DATEN SORTIEREN
280 FOR I=2 TO K
290 N$(0)=N$(I)
300 J=I-1
310 DO WHILE N$(J)>N$(J+1)
320 A$=N$(J+1)
330 N$(J+1) = N$(J)
340 N$(J)=A$
350 J=J-1
```

```
360 LOOP
370 NEXT
380 :
390 :
400 :
410 REM DATEN SORTIERT AUSGEBEN
420 PRINT"NAMEN SORTIERT"
430 PRINT
440 FOR I=1 TO K
450 PRINT N$(I)
460 NEXT
470 END
480 :
490 :
500 :
510 REM DATENWERTE
520 DATA HANS,INGE,FRITZ,RAINER,SIGMUND,ANTON,
                                        SUSANNE
530 DATA ELISABETH,CHRISTA,FRANZ,MARIA,MONIKA,
                                           ENDE
```

Zunächst werden sämtliche Namen, die hier sortiert werden sollen, aus den DATA-Zeilen am Programmende eingelesen und den Elementen des Feldes N$(I) zugeordnet. Zeile 60 dimensioniert das Feld für 20 Einträge. Wir hätten es aber auch ebensogut mit einem anderen Wert dimensionieren können, um Platz für mehr oder weniger Einträge zu schaffen.

Am Ende der DATA-Zeilen steht wieder die Markierung "ENDE", die das Ende der Tabelle kennzeichnet. Wir können nun auch leicht die Anzahl der Werte bestimmen, die die Tabelle enthält. Im Programm ist sie der Variablen K zugeordnet. Ab Zeile 160 werden dann sämtliche Namen in unsortierter Reihenfolge ausgegeben.

Das eigentliche Sortieren des Feldes N$(I) beginnt in Zeile 270. K zeigt auf das letzte Element des Feldes, was besagt, daß alle Elemente von N$(1) bis N$(K) zu sortieren sind. Das ansonsten nicht verwendete Element N$(0) dient hier als Markierung, damit der Tabellenanfang nicht unterlaufen wird.

Bei jedem Sortierschritt wird das Feld N$ in einen bereits sortierten und einen noch unsortierten Teil aufgeteilt. Dabei zeigt die Variable I jeweils auf den ersten Wert des nicht-sortierten Teilfeldes, d.h. die Elemente N$(1)

bis N$(I-1) sind jeweils sortiert, während die Elemente N$(I) bis N$(K) noch nicht sortiert sind.

Der Sortiervorgang beginnt mit dem zweiten Element (FOR...NEXT-Schleife), da eine Tabelle mit einem Element immer sortiert ist. In den Zeilen 310 bis 360 (DO...LOOP-Schleife) wird jeweils das einzufügende Element solange mit den Elementen des sortierten Teils verglichen, bis es an seiner richtigen Stelle steht. Ist es noch größer, als das nächstfolgende Element, wird es mit ihm vertauscht (Zeile 320 bis 340). Damit der Tauschvorgang zwischen den Werten von N$(J) und N$(J+1) stattfinden kann, wird die Hilfsvariable A$ eingeführt. Dieser Vorgang wird nun für jedes Element der Tabelle wiederholt. Dabei nimmt bei jedem Durchgang (FOR...NEXT-Schleife) die Anzahl der sortierten Elemente um eins zu, während die nicht-sortierten Elemente um eins abnehmen. Ist die Tabelle fertig sortiert, werden die Elemente nochmals in ihrer richtigen Reihenfolge ausgegeben (Zeile 410 bis 470).

Das Sortierverfahren durch Einfügen ist nur für kurze Listen geeignet, da bei längeren Listen die Sortierzeit ins Unermeßliche steigt. Das nachfolgend aufgeführte Verfahren nach Shell arbeitet wesentlich schneller, da es nicht zwei nebeneinander liegende Elemente vergleicht und gegebenenfalls vertauscht, sondern zwei Elemente, die weit auseinanderliegen, wobei viele dazwischenliegende Elemente übersprungen werden.

Das Shell-Verfahren ist dem Verfahren durch Einfügen zwar ähnlich, unterscheidet sich aber dadurch, daß anstelle der Elemente N$(J) und N$(J+1) die Elemente N$(J) und N$(J+G) verglichen und (falls nötig) vertauscht werden. Dabei kann G viel größer als 1 sein. Das Shell-Verfahren beinhaltet demgemäß das Einfügverfahren in abgeänderter Form, wobei die Sortiervorgänge für verschiedene Werte von G durchgeführt werden.

Hier nun das gleiche Programm noch einmal mit der Sortierroutine nach Shell:

```
10 REM SORTIEREN NACH SHELL
20 :
30 :
40 :
50 REM DATEN EINLESEN
60 DIM N$(20)
70 I=1
80 DO
```

```
90 READ N$(I)
100 IF N$(I)="ENDE" THEN K=I-1:EXIT
110 I=I+1
120 LOOP
130 :
140 :
150 :
160 REM DATEN UNSORTIERT AUSGEBEN
170 PRINT"NAMEN UNSORTIERT"
180 PRINT
190 FOR I=1 TO K
200 PRINT N$(I)
210 NEXT
220 PRINT
230 PRINT
240 :
250 :
260 :
270 REM DATEN SORTIEREN
280 G=INT(K/2)
290 DO WHILE G>0
300 FOR I=G+1 TO K
310 J=I-G
320 DO WHILE J>0
330 IF N$(J)>N$(J+G) THEN BEGIN
340 A$=N$(J+G)
350 N$(J+G)=N$(J)
360 N$(J)=A$
370 J=J-G
380 BEND:ELSE J=0
390 LOOP
400 NEXT
410 G=INT(G/2)
420 LOOP
430 :
440 :
450 :
460 REM DATEN SORTIERT AUSGEBEN
470 PRINT"NAMEN SORTIERT"
480 PRINT
490 FOR I=1 TO K
500 PRINT N$(I)
```

```
510 NEXT
520 END
530 :
540 :
550 :
560 REM DATENWERTE
570 DATA HANS,INGE,FRITZ,RAINER,SIGMUND,ANTON,
                                          SUSANNE
580 DATA ELISABETH,CHRISTA,FRANZ,MARIA,MONIKA,
                                             ENDE
```

Zu Beginn des Programms wird G auf die halbe Größe der zu sortierenden Liste gesetzt. Danach wird G wiederholt durch 2 geteilt und mit jedem dadurch erhaltenen Wert das abgeänderte Verfahren durch Einfügen durchgeführt. Dies geschieht solange, bis G gleich eins und damit der Sortiervorgang beendet ist.

Versuche haben ergeben, daß das Shell-Verfahren große Datenmengen etwa zehnmal schneller als das Einfügeverfahren sortiert. Bei kleinen Listen mag das Einfügeverfahren geringfügig schneller arbeiten, da es weniger Anweisungen benötigt. Insgesamt betrachtet aber ist das Shell-Verfahren gegenüber dem Verfahren durch Einfügen erheblich vorteilhafter!

6 Dateiverwaltung

In diesem Kapitel geht es um die Dateiverwaltung auf Kassette und Floppy, mit der wir uns nachfolgend ausführlich beschäftigen wollen. Dabei ist es jedoch unerläßlich, daß Sie den Inhalt der vorangegangenen Kapitel gut durchgearbeitet und verstanden haben, da die Dateiverwaltung bereits einige Programmierkenntnisse voraussetzt. Sie sollten, bevor Sie sich der neuen Thematik widmen, zumindest in der Lage sein, ohne große Mühe Ihre eigenen Programme mit Stringvariablen, Stringoperationen und Feldern zu schreiben.

Zunächst werden wir uns nun mit den einzelnen Dateitypen befassen, die es in BASIC gibt. Daran anschließend lernen wir die wichtigsten Anweisungen kennen, die zur Dateiverwaltung in BASIC benötigt werden. Es folgt eine Beschreibung der Dateiverwaltung auf Kassette, die aber nur ein Notbehelf ist und die Möglichkeiten Ihres 128 PC bei weitem nicht ausschöpft.

Interessant wird es dann in den letzten beiden Abschnitten, in denen es um die sequentielle und relative Dateiverwaltung auf der Floppy geht. In diesem Zusammenhang wird ein umfangreiches Adressenverwaltungsprogramm vorgestellt, das vieles beinhaltet und vereinigt, was Sie in diesem Buch bereits kennengelernt haben. Das Programm ist aber gleichzeitig auch als Anregung zur Erstellung eigener Dateiverwaltungsprogramme gedacht.

6.1 Dateitypen

Die Commodore-Floppies sind in der Lage, vier verschiedene Dateitypen zu verarbeiten. Wenn Sie mit Hilfe von

DIRECTORY

das Inhaltsverzeichnis einer Diskette auflisten, erscheint die oberste Zeile in inverser Schrift, die den Diskettennamen, die ID (s. Kapitel 1) und den Vermerk 2A enthält, welcher der Typenkennzeichnung des Disketten-Aufzeichnungsformates entspricht. Unter dieser Kopfzeile werden sämtliche Dateien aufgeführt, die auf der Diskette gespeichert sind.

Die linke Zahl gibt die Anzahl der Blöcke (Sektoren zu je 256 Bytes) an, die die Datei auf der Diskette belegt. Dann folgt der Dateiname und rechts der Dateityp, der mit drei Buchstaben gekennzeichnet ist. Dabei bedeuten

PGR = Programmdatei
SEQ = sequentielle Datei
REL = relative Datei
USR = Anwender- (User-) Datei

Programmdateien lernten wir bereits im ersten Kapitel kennen, auch wenn wir sie dort noch nicht mit diesem Namen bezeichnet haben. Programme sind im Sinne der Datenspeicherung auch Dateien, die jedoch nicht aus Datenposten, wie z.B. Artikelnummer, Artikelbeschreibung oder Preis bestehen, sondern aus einem Programmtext, den wir durch LIST sichtbar machen können. Programme sind allerdings nur teilweise im Klartext, d.h. in reinem ASCII-Code gespeichert, während ein Großteil aus Binärcodes und Tokens (binäre Kurzform für einzelne BASIC-Anweisungen) besteht.

Da wir den Umgang mit Programmdateien, insbesondere die Anweisungen zum Speichern und Laden von Programmen wie LOAD, DLOAD, SAVE und DSAVE, ausführlich im ersten Kapitel behandelt haben, wollen wir auf eine Wiederholung an dieser Stelle verzichten. Ebenfalls in Kapitel 1 haben wir Diskettenbefehle kennengelernt, die auch für alle anderen Dateitypen gelten und deshalb nochmals kurz erwähnt werden sollen:

HEADER	"Dateiname, ID"	Formatieren einer Diskette
SCRATCH	"Dateiname"	Löschen einer Datei
RENAME	"Alter Name" TO "Neuer Name"	Umbenennen einer Datei

Beachten Sie bitte, daß bei SCRATCH und RENAME nur der Dateiname angegeben werden muß, nicht aber der Dateityp. Deshalb ist es auch unzulässig, auf einer Diskette zwei Dateien unter gleichem Namen aber von unterschiedlichem Typ abzuspeichern.

Mit sequentiellen und relativen Dateien werden wir uns in diesem Kapitel noch ausführlich beschäftigen. Sequentielle Dateien bestehen aus Datensätzen, die hintereinander (sequentiell) im ASCII-Code auf Diskette geschrieben und ebenso wieder gelesen werden. Wenn Sie also den 100. Datensatz einer sequentiellen Datei benötigen, müssen Sie die 99 davorliegenden ebenfalls einlesen, was unter Umständen eine sehr zeitraubende Angelegenheit ist.

Dies ist bei relativen Dateien anders, denn hier werden auf der Diskette sogenannte Records mit fester Länge angelegt, auf die dann direkt zugegriffen werden kann, ohne daß alle anderen Records zuvor gelesen werden müssen. Das Diskettenbetriebssystem (DOS) reserviert dabei automatisch genügend Speicherplatz und führt eine interne Tabelle, die angibt, in welchem Sektor und an welcher Stelle sich darin der betreffende Record befindet.

Unter den letzten Datetyp, der von der Floppy verwaltet werden kann, fallen die Anwender- oder User-Dateien. Diese Art von Datei dient nur besonderen Zwecken, die sich meist auf die Maschinenprogrammierung des DOS oder des Betriebssystems im Computer beziehen. Da wir aber im vorliegenden Buch die BASIC-Programmiersprache erlernen wollen, benötigen wir diesen Dateityp nicht und wollen deshalb auch nicht näher darauf eingehen.

6.2 BASIC-Anweisungen zur Dateiverwaltung

Jedesmal, bevor ein Zugriff auf ein Peripheriegerät (Drucker, Floppy, Kassettenrecorder) vorgenommen wird, muß die Verbindung zu dem betreffenden Gerät aktiviert werden. Wenn Sie Programme laden oder abspeichern, geschieht dies automatisch, indem Sie LOAD (DLOAD) bzw. SAVE (DSAVE) eingeben. Das Betriebssystem öffnet dann eine Programmdatei, schreibt das Programm auf Diskette oder Band und schließt die Datei wieder.

Nicht ganz so bequem ist das Floppy-Handling für sequentielle oder relative Dateien. Hier muß das Öffnen, die Aus-/Eingabe und das Schließen separat vorgenommen werden. Das gleiche gilt im Prinzip auch für den Drucker, obwohl Sie an ihn nur Daten ausgeben können, aber es nicht möglich ist, Daten davon einzulesen.

Das wichtigste ist das richtige Öffnen des Ein-/Ausgabekanals zu dem betreffenden Gerät, was mit der OPEN-Anweisung geschieht.
Hier ihre allgemeine Form:

```
OPEN lf, dn, sa, "name"
```

Darin bedeuten

lf	= logische Filenummer
dn	= Gerätenummer
sa	= Sekundäradresse
name	= Dateiname

File kommt aus dem Amerikanischen und heißt übersetzt Datei. Ein logisches File ist also eine logische Datei, die keinesfalls mit unserer auszugebenden oder einzulesenden Datei identisch ist. In jeder OPEN-Anweisung muß ein logisches File definiert werden, wodurch BASIC einen bestimmten Übertragungskanal zu einem Gerät herstellt. So können Sie mehrere (bis zu 10) logische Files gleichzeitig öffnen, wobei z.B. das File mit der Nummer 1 einem Eingabekanal von der Floppy entspricht, das File mit der Nummer 2 einem Ausgabekanal zur Floppy und das File mit der Nummer 3 einem Ausgabekanal zum Drucker.

In der Wahl der Filenummer sind Sie weitestgehend frei, so daß Sie den gerade genannten Kanälen auch die Filenummern 25, 61 und 103 geben könnten. Filenummern können beliebige Werte zwischen 1 und 255 annehmen, wobei diejenigen, die größer als 127 sind, jedoch eine Sonderfunktion haben. Wird nämlich am Ende eines Datensatzes ein Carriage Return oder CHR$(13) gesendet, erzeugen Files mit höheren Nummern zusätzlich noch ein Line Feed bzw. CHR$(10). Dies ist jedoch nur bei der Übertragung zu anderen Rechnern von Bedeutung, da Commodore-Computer in der Regel ohne Line Feed auskommen.

Sie können auch zur Floppy mehrere logische Files gleichzeitig öffnen, was bei sehr umfangreichen Dateiverwaltungen durchaus vorkommen kann. In unserem Beispiel waren es immerhin zwei Files, die einen Schreib- und einen Lesekanal verwalteten.

Das nächste, was in der OPEN-Anweisung angegeben werden muß, ist die Gerätenummer, die sich theoretisch zwischen 1 und 15 bewegen kann. Normalerweise werden aber nur folgende Nummern benötigt:

Nummer	Gerät
0	Tastatur
1	Kassettenrecorder
2	RS-232-Schnittstelle
3	Bildschirm
4	Drucker
5	eventuell zweiter Drucker oder Plotter
8	Floppylaufwerk
9	eventuell zweites Floppylaufwerk

Uns interessieren hier nur die Gerätenummern 1 für die Datasette, 4 für den Drucker und 8 für die Floppy. Die RS-232-Schnittstelle benötigen Sie nur zur Datenfernübertragung, wenn Sie z.B. ein Modem anschließen. Vielleicht sind Sie darüber erstaunt, daß auch die Tastatur und der Bildschirm eine Gerätenummer besitzen. Diese Nummern müssen normalerweise jedoch nicht angegeben werden, da sie durch das Betriebssystem automatisch gesetzt werden, wenn sie z.B. eine Taste drücken oder mit PRINT etwas auf den Bildschirm ausgeben; nur in wenigen Ausnahmefällen ist die Angabe dieser Gerätenummern erforderlich.

Der dritte Parameter in der OPEN-Anweisung ist die Sekundäradresse, die von Gerät zu Gerät eine unterschiedliche Bedeutung haben kann. So geben z.B. die meisten Drucker Groß- und Kleinbuchstaben aus, wenn ihnen zuvor die Zahl 7 als Sekundäradresse übermittelt wurde. Eine andere Sekundäradresse steuert z.B. den Graphikmodus oder den Zeilenabstand des Druckers, während er für den Normalbetrieb keine oder die Sekundäradresse 0 benötigt.

Als letztes folgt noch der Dateiname der (richtigen!) Datei. Er kann in manchen Fällen noch weitere Parameter enthalten, wie wir bei den Floppyoperationen noch sehen werden.

Nicht immer muß eine OPEN-Anweisung sämtliche Parameter enthalten. So genügt es z.B. beim Drucker meistens, wenn Sie nur die Filenummer und die Gerätenummer angeben.

Um Verwechslungen zwischen den logischen Dateien (Files) und den "echten" Dateien zu vermeiden, wollen wir im folgenden die logischen Files mit "Files" bezeichnen und für die zu übertragenden Dateien das deutsche Wort "Datei" belassen.

Nachdem nun ein File eröffnet ist, können auf ein Gerät Daten ausgegeben oder von ihm eingelesen werden. Ob es sich um einen Aus- oder Eingabekanal handelt, ist meist in der Sekundäradresse oder im Dateinamen festgelegt. Zur Ausgabe in ein eröffnetes File dient die PRINT#-Anweisung, zum Einlesen entweder die INPUT#- oder die GET#-Anweisung. PRINT# gibt einen ASCII-String aus, der mit einem Carriage Return abgeschlossen sein muß. Ist eine Zahl oder Fließkommavariable angegeben, wird diese zunächst in einen String umgewandelt (vgl. STR$-Anweisung) und dann ausgegeben. INPUT# bewirkt das Gegenteil von PRINT#; es liest einen String ein. Einen Sonderfall stellt die GET#-Anweisung dar, die nur zum Lesen verwendet werden kann und immer nur ein Zeichen einliest. Hier einige Beispiele:

PRINT#2,"ABC"	gibt den String "ABC" auf den mit der logischen Filenummer 2 zugeordneten Kanal aus.
PRINT#2, A$	dasselbe geschieht mit dem Inhalt von A$
PRINT#2, C	dasselbe geschieht mit dem Inhalt der numerischen Variablen C, nachdem er in ASCII-Zeichen umgeformt wurde.
INPUT#3, C$	liest einen String von dem Gerät, welchem das logische File 3 zugeordnet wurde und legt ihn in der Variablen C$ ab.
GET#1, A$	liest ein Zeichen aus dem logischen File 1 und ordnet es A$ zu.

Wir sehen also, daß in den Ein-/Ausgabe-Anweisungen immer nur die logische Filenummer angegeben werden muß, die sich auf die entsprechende OPEN-Anweisung bezieht. Wurde ein File nicht eröffnet, erscheint die Fehlermeldung

?FILE NOT OPEN ERROR IN (Zeilennummer)

Nach Abschluß aller Ein-/Ausgabeoperationen dürfen Sie nicht vergessen, das logische File zu schließen. Dies erfolgt mit der CLOSE-Anweisung. So schließt

```
CLOSE 2
```

den Ein- oder Ausgabekanal für das Gerät, das ihm in der entsprechenden OPEN-Anweisung zugeordnet wurde. Wenn Sie beim Drucker vergessen, das logische File zu schließen, sind die Folgen noch relativ harmlos. Schlimmer ist es dagegen, wenn Sie Daten auf die Floppy geschrieben haben, die dann unter Umständen teilweise verlorengehen oder nur unter großen Schwierigkeiten wieder gelesen werden können.

Für Floppyoperationen hat BASIC 7.0 die DOPEN- /DCLOSE-Anweisung.

```
DOPEN#1,"TEST",W
```

eröffnet eine sequentielle Datei unter dem Namen "TEST" zum sequentiellen Schreiben unter der Filenummer 1. Dabei wird die Gerätenummer und die Sekundäradresse automatisch gesetzt und übermittelt.

```
DOPEN#1,"TEST"
```

eröffnet die gleiche Datei zum Lesen. Dabei fehlt nur der durch Komma abgetrennte Zusatz W für Schreiben. Genauso wird diese Datei in beiden Fällen mit

```
DCLOSE#1
```

wieder geschlossen.

Abschließend wollen wir noch die Anweisung CMD betrachten, die sich für einige Fälle als recht nützlich erweist. CMD leitet nämlich die Bildschirmausgabe auf ein anderes Peripheriegerät um. Leider enthält BASIC 7.0 keine Anweisung zur Ausgabe von Programmlistings auf dem Drucker, so daß wir einen Trick anwenden müssen, um dies dennoch zu ermöglichen:

```
OPEN 1,4
CMD 1
LIST
PRINT#1
CLOSE 1
```

Da die LIST-Anweisung normalerweise nur auf dem Bildschirm funktioniert, leiten wir die Bildschirmausgabe vorher auf den Drucker um. Für die Ausgabe muß aber zuerst ein logisches File eröffnet werden (OPEN 1,4). CMD 1 dirigiert dann die Bildschirmausgabe auf den Drucker und LIST listet das Programm. Im Anschluß wird durch ein einfaches PRINT#1 die Ausgabe wieder zurück auf den Bildschirm gelegt, worauf CLOSE 1 das logische File 1 wieder schließt.

Dasselbe, was wir gerade mit dem Drucker gemacht haben, können wir auch mit jedem anderen Ausgabegerät praktizieren. Durch geringe Abwandlung der obigen Anweisungen leiten wir diesmal die Ausgabe auf die Floppy um und erzeugen eine sequentielle Datei. Diese Methode, die Ihnen den Programmtext in reinem ASCII-Code liefert, ist besonders dann hilfreich, wenn Sie das Listing in ein Textsystem übernehmen möchten, das mit sequentiellen Dateien arbeitet.

```
OPEN 2,8,2,"ABC,S,W"
CMD 2
LIST
PRINT#2
CLOSE 2
```

Nach Ausführung dieser Anweisungen befindet sich nun Ihr Programmlisting in der sequentiellen Datei ABC, die Sie genauso wie jede andere sequentielle Datei wieder lesen können.

6.3 Dateiverwaltung auf Kassette

Auf Kassette kann aus verständlichen Gründen nur eine sequetielle Dateiverwaltung durchgeführt werden, da das Band keinen Direktzugriff erlaubt. Hier nun ein Beispiel zum Schreiben und Lesen von Daten auf Kassette:

```
10 REM DATEN AUF KASSETTE SCHREIBEN
20 OPEN 1,1,1,"TEST"
30 DO
40 READ A$
50 IF A$ ="ENDE" THEN EXIT
60 PRINT#1, A$
70 LOOP
80 CLOSE 1
90 DATA WURST, KAESE, EIER, FLEISCH, GEMUESE
100 DATA OBST, HERING, MEHL, ROSINEN, ENDE
```

Programme mit ähnlichem Aufbau von READ und DATA haben wir bereits schon früher kennengelernt. Neu ist nur, daß diesmal die Ausgabe auf den Kassettenrecorder geleitet wird.

Zunächst wird das logische File 1 geöffnet, das einen Ausgabekanal (Sekundäradresse 1) auf die Datasette (Gerätenummer 1) legt. Daraufhin werden sämtliche Werte aus den DATA-Zeilen gelesen und mit PRINT#1 ausgegeben. Der String "ENDE" kennzeichnet auch hier wieder den letzten Wert, so daß nach übertragung sämtlicher Strings die DO...LOOP-Schleife verlassen und die CLOSE-Anweisung ausgeführt wird.

Nach Ausführung der OPEN-Anweisung erscheint

```
PRESS PLAY & RECORD ON TAPE
```

auf dem Bildschirm, so daß Sie erst die RECORD- und PLAY-Taste am Recorder drücken müssen, damit die Daten aufgezeichnet werden können. Während sämtlicher Kassettenoperationen bleibt der Bildschirm dunkel, was im Betriebssystem begründet ist. Ist die Aufzeichnung abgeschlossen, spulen Sie das Band am besten gleich wieder zurück.

Nun wollen wir ein Programm schreiben, das die aufgezeichneten Daten wieder von Kassette liest. Hier das Listing:

```
10 REM DATEN VON KASSETTE LESEN
20 OPEN 2,1,0,"TEST"
30 INPUT#2, A$
40 PRINT A$
50 IF ST=0 THEN 30
60 CLOSE 2
```

Auch hier erscheint zunächst die Meldung

PRESS PLAY ON TAPE

die Sie auffordert, die Play-Taste zu drücken. Zeile 20 öffnet das File 2 zum Lesen von Kassette, was sich durch die Sekundäradresse 0 ausdrückt. Daraufhin liest Zeile 30 jeden einzelnen Datenwert mit der INPUT#-Anweisung ein und ordnet ihn der Variablen A$ zu. Zeile 40 gibt die eingegebenen Werte zur Kontrolle auf dem Bildschirm aus. Zeile 50 fragt die Statusvariable ST ab. Diese reservierte Variable gibt an, ob der Datentransfer einwandfrei funktioniert, dann nämlich ist ihr Wert 0. Ist der letzte Wert eingelesen, also das Dateiende erreicht, erhält sie den Wert 64, also ungleich Null. Dann läuft das Programm zu Zeile 60 weiter, wo das logische File 2 geschlossen wird.

6.4 Sequentielle Dateiverwaltung auf Diskette

Dasselbe, was wir gerade mit der Kassette durchgeführt haben, gilt im Prinzip auch für das Floppylaufwerk. Der einzige Unterschied liegt in der OPEN-Anweisung und in der Möglichkeit, Fehlermeldungen des DOS abzufragen. Nachfolgend noch einmal dasselbe Programm für den Diskettenbetrieb:

```
10 REM DATEN AUF DISKETTE SCHREIBEN
20 DOPEN#1,"TEST",W
25 IF DS<>0 THEN PRINT DS$ : GOTO 80
30 DO
40 READ A$
50 IF A$ ="ENDE" THEN EXIT
60 PRINT#1, A$
70 LOOP
80 DCLOSE#1
90 DATA WURST, KAESE, EIER, FLEISCH, GEMUESE
100 DATA OBST, HERING, MEHL, ROSINEN, ENDE
```

Wir verwenden hier in Zeile 20 die bequeme DOPEN-Anweisung, die sich auf das logische File 1 bezieht. Die neu eingefügte Zeile 25 prüft, ob eine Fehlermeldung des DOS vorliegt. In diesem Fall ist der Wert der reservierten Variablen DS ungleich Null und es erscheint die dazugehörige Fehlermeldung in der ebenfalls reservierten Variablen DS$. Ein Fehler kann z.B. auftreten, wenn eine Datei unter gleichem Namen bereits auf der Diskette existiert oder die Diskette schreibgeschützt (Kerbe am rechten Rand überklebt) ist. In einem solchen Fall wird gar nicht erst versucht, Daten auf die Diskette zu schreiben, sondern sofort die DCLOSEAnweisung in Zeile 80 ausgeführt. Das restliche Programm arbeitet genauso wie beim Kassettenbetrieb.

Zum Einlesen der Daten von Diskette wollen wir ein Programm verwenden, das die Daten nicht nur auf dem Bildschirm, sondern zusätzlich noch auf dem Drucker ausgibt. Es könnte z.B. so aussehen:

```
10 REM DATEN VON DISKETTE LESEN UND AUSDRUCKEN
20 DOPEN#2,"TEST"
30 IF DS<>0 THEN PRINT DS$: GOTO 100
40 OPEN 4,4
50 INPUT#2, A$
60 PRINT A$
70 PRINT#4, A$
80 IF ST=0 THEN 50
90 CLOSE 4
100 DCLOSE#2
```

Zeile 20 eröffnet die sequentielle Diskettendatei "TEST", diesmal aber zum Lesen, d.h. ohne durch ein Komma abgetrenntes W. Wie beim Schreiben wird auch hier in Zeile 30 der Fehlerkanal des DOS abgefragt und im Falle eines Fehlers die Fehlermeldung ausgegeben und daraufhin das Programm abgebrochen. Zeile 40 führt eine OPENAnweisung zur Druckerausgabe durch. Da wir keinen besonderen Druckmodus wünschen, können wir in diesem Fall auf die Sekundäradresse und ohnehin auf den Dateinamen verzichten, denn der Drucker nimmt keine Dateinamen an. Zeile 50 liest die Daten von der Floppy und legt sie zunächst in A$ ab. Zeile 60 gibt sie dann auf dem Bildschirm und Zeile 70 auf dem Drucker aus. Zeile 80 überprüft wieder anhand der Statusvariablen ST das Dateiende. Ist dies noch nicht erreicht, springt das Programm zurück in Zeile 50, um den nächsten Datenwert einzulesen und auszugeben. Schließlich wird nach Erreichen des Dateiendes in Zeile 90 das logische File 4 für den Drucker und in Zeile 100 das File 2 für die Floppy wieder geschlossen.

Zum Öffnen und Schließen der Diskettendatei haben wir hier die vereinfachten Anweisungen DOPEN und DCLOSE verwendet. Leser, die bereits früher mit dem C-64 gearbeitet haben, wissen, daß man auch mit den normalen OPEN- und CLOSE-Anweisungen arbeiten kann. Der Vollständigkeit halber wollen wir sie kurz betrachten.

Zum Öffnen einer sequentiellen Ausgabedatei auf Floppy dient folgende OPEN-Anweisung:

```
OPEN 1,8,2,"TEST,S,W"
```

Soll die Datei zum Lesen geöffnet werden, ist folgendes einzugeben:

```
OPEN 2,8,2,"TEST,S,R"
```

Die Sekundäradresse ist für unsere Zwecke unerheblich und kann hier zwischen 2 und 14 liegen.

Geschlossen wird die Datei ganz normal mit CLOSE1 bzw. CLOSE2. Sie sehen also, daß bei Floppy-Operationen der Dateiname angibt, ob gelesen oder geschrieben werden soll. Hinter dem eigentlichen Dateinamen steht nochmals ‚S,W bzw. ‚S,R. Das S gibt an, daß eine sequentielle Datei eröffnet werden soll, W steht für Schreiben und R für Lesen.

Erfahrene Profis wissen vielleicht, daß man auf diese Weise auch ein normal mit SAVE bzw. DSAVE abgespeichertes BASIC-Programm "lesen" kann und eröffnen die Programmdatei folgendermaßen:

```
OPEN2,8,2,"ABC,P,R"
```

Hinter dem eigentlichen Dateinamen, der in unserem Beispiel "ABC" lautet, steht ‚P,R, wobei das P eine Programmdatei angibt, die "gelesen" werden soll. Mit "Lesen" ist jedoch nicht das einfache Einlesen mit INPUT# und die Ausgabe auf dem Bildschirm mit PRINT gemeint. Sollten Sie das versuchen, werden Sie wahrscheinlich die Erfahrung machen, daß Ihr Bildschirm mit lauter undefinierbaren Zeichen übersät ist. Aber Sie wissen ja bereits, daß Programmdateien zum Teil aus Binärwerten bestehen, die - sollten sie fälschlicherweise als ASCII-Zeichen interpretiert werden - dieses Wirrwarr verursachen. Deshalb sind solche Spielereien nur dem versierten Profi zu empfehlen, der auch weiß, wie er mit den eingelesenen Zeichen umzugehen hat.

6.4.1 Adressenverwaltungsprogramm (sequentiell)

Als ausführliches Beispiel für die sequentielle Dateiverwaltung auf Diskette wollen wir hier ein komplettes Adressenverwaltungsprogramm betrachten. Dieses Programm ist modular aufgebaut und menügesteuert. Es kann in leicht abgeänderter Form auch für jede andere Art von Dateiverwaltung eingesetzt werden.

Nach dem Laden und Starten erscheint zunächst das Menü, das folgendermaßen aussieht:

```
                ADRESSENVERWALTUNG
                ==================

    ADRESSEN SCHREIBEN/FORTSCHREIBEN          1

    ADRESSEN AENDERN/LOESCHEN                 2

    ADRESSEN SORTIEREN                        3

    ADRESSEN LADEN                            4

    ADRESSEN ABSPEICHERN                      5

    ADRESSEN AUSDRUCKEN                       6

    ADRESSEN AUF ETIKETTEN AUSDRUCKEN         7


    E N D E                                   E


    BITTE WAEHLEN SIE
```

Wie Sie sehen, bietet dieses Programm verschiedene Möglichkeiten zur Datenbehandlung. Menüpunkt 1 dient zum Schreiben und Fortschreiben der Adressendatei. Schreiben bedeutet, daß Sie eine Datei, die vorher leer war,

von Anfang an mit Daten versehen. Mit Fortschreiben ist gemeint, daß eine Datei, die bereits angelegt wurde, nun erweitert werden soll. Die Datei kann vorher bereits auf Diskette abgespeichert gewesen sein oder gerade geschrieben und sortiert oder ausgedruckt werden usw.

Wenn Sie nun Menüpunkt 1 auswählen, werden Sie aufgefordert, verschiedene Daten einzugeben:

```
          ADRESSEN SCHREIBEN/FORTSCHREIBEN

12

ANREDE (0-5)  ? 1

NAME          ? MEIER

VORNAME       ? FRANZ

STRASSE       ? BERGSTR. 47

PLZ/ORT       ? 8000 MUENCHEN 80

NAECHSTE EINGABE - RETURN-TASTE DRUECKEN

ZURUECK INS MENUE - LEERTASTE DRUECKEN
```

Dies ist ein komplettes Beispiel, wie die Daten eingegeben werden müssen. Direkt unter der Überschrift befindet sich eine Zahl, die angibt, den wievielten Datensatz wir gerade bearbeiten. In diesem Fall ist es der zwölfte. Eine bisher leere Datei beginnt immer mit dem 1. Datensatz, während eine fortgeschriebene, die z.B. von der Diskette eingelesen wurde, mit dem nächstfolgenden Datensatz fortfährt. Lesen Sie z.B. eine Datei mit 58 Datensätzen von der Diskette ein, erscheint beim Fortschreiben oben die Nummer 59. Die Datensätze werden zunächst in der Reihenfolge ihrer Eingabe abgelegt, ungeachtet der Tatsache, ob sie sich in alphabetischer Reihenfolge befinden oder nicht.

Bevor Sie die eigentlichen Daten eingeben, erscheint unter der fortlaufenden Nummer zunächst nur die Aufforderung, den Code für die Anrede

einzugeben, während der Rest des Bildschirms noch leer bleibt. Bei der Anrede können Sie unter sechs Möglichkeiten auswählen:

Code	ANREDE
0	keine
1	HERRN
2	FRAU
3	FRAEULEIN
4	HERRN UND FRAU
5	FIRMA

Diese Codes sollten Sie sich vorher merken oder notieren, da sie einfacher einzugeben sind, als wenn Sie jedesmal die Anrede ausschreiben. Nach dem Drücken der RETURN-Taste erscheint als nächstes die Aufforderung zur Eingabe des (Nach-)Namens. Sie geben ihn ein und drücken wiederum die RETURN-Taste. Das gleiche wiederholt sich dann für die Straße und den Ort mit Postleitzahl.

Ist ein Datensatz komplett eingegeben, erhalten Sie unten im Bildschirm den Hinweis, entweder die RETURN- oder die Leertaste zu drücken. Letztere führt Sie wieder zurück ins Menü, während Sie mit der RETURN-Taste gleich zum nächsten Datensatz kommen. Der Vorgang wiederholt sich danach von vorne, allerdings mit einer um eins erhöhten Datensatznummer.

Achten Sie darauf, daß die einzelnen Eingabewerte eine bestimmte Anzahl von Zeichen nicht überschreiten dürfen:

Name	15 Zeichen
Vorname	15 Zeichen
Straße	20 Zeichen
PLZ/Ort	21 Zeichen

Kommen wir jetzt zum 2. Menüpunkt, der zum Ändern und Löschen von Adressen dient. Sie werden zunächst aufgefordert, den betreffenden Namen als Suchindex einzugeben. Daraufhin sucht das Programm, ob dieser Name in der Datei vorhanden ist. Wenn nein, erscheint ein Hinweis "NICHT GEFUNDEN" und Sie können den nächsten Namen suchen oder ins Menü zurückkehren. Wurde der Name dagegen gefunden, erscheint der komplette Datensatz auf dem Bildschirm, ähnlich wie unter Menüpunkt 1, wobei der Cursor über dem Anredecode steht. Sie können nun die Daten ändern und

abschließend die RETURN-Taste drücken. Wenn Sie keine Änderung vornehmen, drücken Sie jeweils nur die RETURN-Taste. Nach Beendigung des Korrekturvorgangs können Sie, wie in Punkt 1, entweder fortfahren (RETURN-Taste) oder ins Menü zurückkehren.

Soll ein Datensatz gelöscht werden, wählen Sie ebenfalls Menüpunkt 2 und überschreiben den Anredecode mit dem Buchstaben L. Dann drücken Sie die RETURN-Taste, worauf der betreffende Datensatz aus der Datei entfernt wird. Durch den Löschvorgang rücken alle Datensätze mit höherer Nummer um eins nach unten, wodurch ihre Gesamtzahl um eins abnimmt.

Menüpunkt 3 sortiert die Datensätze in alphabetischer Reihenfolge nach dem Shell-Sortierverfahren, das wir bereits besprochen haben. Die Nummern der Datensätze ändern sich dadurch entsprechend.

Wir kommen nun zu den reinen Floppy-Operationen (Menüpunkt 4 und 5). In beiden Fällen werden Sie aufgefordert, den Dateinamen einzugeben, unter dem die Adressen auf Diskette abgelegt sind bzw. abgelegt werden sollen. Das Programm prüft zunächst automatisch, ob die Floppy eingeschaltet ist, wenn nicht, werden Sie aufgefordert, dies zu tun.

Funktioniert ein Schreib- bzw. Lesevorgang auf der Floppy nicht einwandfrei, erscheinen die betreffenden Original-Fehlermeldungen des Disketten-Betriebssystems (DOS) auf dem Bildschirm. Existiert beim Schreibvorgang bereits eine gleichnamige Datei, werden Sie gefragt, ob diese überschrieben werden darf.

Die letzten beiden Menüpunkte dienen zum Ausdrucken der Datei. Punkt 6 liefert einen normalen Listenausdruck, während Sie mit Punkt 7 Etiketten beschriften können. Dabei wird davon ausgegangen, daß die Etiketten genau 9 Druckzeilen auseinander liegen. Sollten Sie Etiketten anderer Größe verwenden, können Sie das entsprechende Teilprogramm ab Zeile 2940 durch Hinzufügen oder Entfernen von PRINT#4-Anweisungen leicht anpassen. Bei beiden Menüpunkten wird zunächst geprüft, ob der Drucker eingeschaltet ist. Wenn nicht, werden Sie aufgefordert, dies nachzuholen. Nachfolgend das komplette Programmlisting:

```
10 REM ADRESSENVERWALTUNGSPROGRAMM (SEQUENTIELL)
20 REM========================================
30 REM
40 REM
50 MA=200:REM MAX.ANZAHL DER DATENSAETZE
60 K=0:REM DATENSAETZE IM SPEICHER
70 LN=15:REM ZEICHEN LAENGE NAMEN
80 LV=15:REM ZEICHEN LAENGE VORNAMEN
90 LS=20:REM ZEICHEN LAENGE STRASSE
100 LO=21:REM ZEICHEN LAENGE PLZ/ORT
110 LA=1:REM ZEICHEN LAENGE ANREDECODE
120 N$="                    "
130 DIM N$(MA)
140 DIM AR$(5)
150 AR$(0)=""
160 AR$(1)="HERRN"
170 AR$(2)="FRAU"
180 AR$(3)="FRAEULEIN"
190 AR$(4)="HERRN UND FRAU"
200 AR$(5)="FIRMA"
210 :
220 :
230 :
240 REM MENUE
250 PRINT"<CLR>"
260 PRINT TAB(10);"ADRESSENVERWALTUNG"
270 PRINT TAB(10);"=================="
280 PRINT
290 PRINT
300 PRINT"ADRESSEN SCHREIBEN/FORTSCHREIBEN       1"
310 PRINT
320 PRINT"ADRESSEN AENDERN/LOESCHEN              2"
330 PRINT
340 PRINT"ADRESSEN SORTIEREN                     3"
350 PRINT
360 PRINT"ADRESSEN LADEN                         4"
370 PRINT
380 PRINT"ADRESSEN ABSPEICHERN                   5"
390 PRINT
400 PRINT"ADRESSEN AUSDRUCKEN                    6"
410 PRINT
420 PRINT"ADRESSEN AUF ETIKETTEN AUSDRUCKEN      7"
```

```
430 PRINT
440 PRINT
450 PRINT"E N D E                                          E"
460 PRINT
470 PRINT
480 PRINT"BITTE WAEHLEN SIE"
490 GETKEY M$
500 IF M$="E" THEN END
510 M=VAL(M$)
520 IF M<1 OR M>7 THEN 490
530 ON M GOSUB 580,980,1620,1930,2210,2590,2940
540 GOTO 240
550 :
560 :
570 :
580 REM ADRESSEN SCHREIBEN/FORTSCHREIBEN
590 PRINT"<CLR>"
600 PRINT
610 PRINT
620 PRINT"    ADRESSEN SCHREIBEN/FORTSCHREIBEN    "
630 PRINT
640 PRINT
650 K=K+1
660 PRINT K
670 PRINT
680 IF K>MA THEN PRINT"DATEI VOLL":GOTO 850
690 PRINT"ANREDE (0-5)";:INPUTA$
700 A=VAL(A$)
710 IF A<0 OR A>5 OR LEN(A$)<>1 THEN PRINT"<2xCRSR UP>"
                                                    :GOTO 690
720 PRINT
730 PRINT"NAME        ";:INPUT NA$
740 PRINT
750 NA$=LEFT$(NA$+N$,LN)
760 PRINT"VORNAME     ";:INPUT NV$
770 PRINT
780 NV$=LEFT$(NV$+N$,LV)
790 PRINT"STRASSE     ";:INPUT NS$
800 PRINT
810 NS$=LEFT$(NS$+N$,LS)
820 PRINT"PLZ, ORT    ";:INPUT NO$
830 NO$=LEFT$(NO$+N$,LO)
```

```
840 N$(K)=NA$+NV$+NS$+NO$+RIGHT$(STR$(A),1)
850 PRINT
860 PRINT
870 PRINT"NAECHSTE EINGABE - RETURN-TASTE DRUECKEN"
880 PRINT
890 PRINT"ZURUECK INS MENUE - LEERTASTE DRUECKEN"
900 GETKEY M$
910 IF M$=CHR$(13) THEN 580
920 IF M$=" " THEN 940
930 GOTO 900
940 RETURN
950 :
960 :
970 :
980 REM ADRESSEN AENDERN/LOESCHEN
990 PRINT"<CLR>"
1000 PRINT
1010 PRINT
1020 PRINT"        ADRESSEN AENDERN/LOESCHEN"
1030 PRINT
1040 PRINT
1050 PRINT"NAME          ";:INPUT NA$
1060 NA$=LEFT$(NA$+N$,LN)
1070 I=1
1080 DO UNTIL I>K
1090 IF LEFT$(N$(I),LN)=NA$ THEN 1150
1100 I=I+1
1110 LOOP
1120 PRINT
1130 PRINT"NICHT GEFUNDEN"
1140 GOTO 1490
1150 NA$=LEFT$(N$(I),LN)
1160 NV$=MID$(N$(I),LN+1,LV)
1170 NS$=MID$(N$(I),LN+LV+1,LS)
1180 NO$=MID$(N$(I),LN+LV+LS+1,LO)
1190 A$=RIGHT$(N$(I),1)
1200 PRINT I
1210 PRINT
1220 PRINT"ANREDE (0-5)  ";A$
1230 PRINT
1240 PRINT"NAME           ";NA$
1250 PRINT
```

```
1260 PRINT"VORNAME          ";NV$
1270 PRINT
1280 PRINT"STRASSE         ";NS$
1290 PRINT
1300 PRINT"PLZ, ORT        ";NO$
1310 PRINT"<10xCRSR UP>"
1320 PRINT"<12xCRSR RIGHT>";:INPUT A$
1330 A=VAL(A$)
1340 IF A$="L" THEN GOSUB 3250:GOTO 1490
1350 IF A<0 OR A>5 OR LEN(A$)<>1 THEN PRINT"<2xCRSR UP>"
                                                         :GOTO 1320
1360 PRINT
1370 PRINT"<12xCRSR RIGHT>";:INPUT NA$
1380 PRINT
1390 NA$=LEFT$(NA$+N$,LN)
1400 PRINT"<12xCRSR RIGHT>";:INPUT NV$
1410 PRINT
1420 NV$=LEFT$(NV$+N$,LV)
1430 PRINT"<12xCRSR RIGHT>";:INPUT NS$
1440 PRINT
1450 NS$=LEFT$(NS$+N$,LS)
1460 PRINT"<12xCRSR RIGHT>";:INPUT NO$
1470 NO$=LEFT$(NO$+N$,LO)
1480 N$(I)=NA$+NV$+NS$+NO$+RIGHT$(STR$(A),1)
1490 PRINT
1500 PRINT
1510 PRINT"NAECHSTE EINGABE - RETURN-TASTE DRUECKEN"
1520 PRINT
1530 PRINT"ZURUECK INS MENUE - LEERTASTE DRUECKEN"
1540 GETKEY M$
1550 IF M$=CHR$(13) THEN 980
1560 IF M$=" " THEN 1580
1570 GOTO 1540
1580 RETURN
1590 :
1600 :
1610 :
1620 REM ADRESSEN SORTIEREN
1630 PRINT"<CLR>"
1640 PRINT
1650 PRINT
1660 PRINT"          ADRESSEN SORTIEREN"
```

```
1670 PRINT
1680 PRINT
1690 G=INT(K/2)
1700 DO WHILE G>0
1710 FOR I=G+1 TO K
1720 J=I-G
1730 DO WHILE J>0
1740 IF N$(J)>N$(J+G) THEN BEGIN
1750 A$=N$(J+G)
1760 N$(J+G)=N$(J)
1770 N$(J)=A$
1780 J=J-G
1790 BEND:ELSE J=0
1800 LOOP
1810 NEXT
1820 G=INT(G/2)
1830 LOOP
1840 PRINT
1850 PRINT
1860 PRINT"          ZURUECK INS MENUE"
1870 PRINT"        BELIEBIGE TASTE DRUECKEN"
1880 GETKEY M$
1890 RETURN
1900 :
1910 :
1920 :
1930 REM ADRESSENDATEI LADEN
1940 GOSUB 3550
1950 PRINT"<CLR>"
1960 PRINT
1970 PRINT
1980 PRINT"          ADRESSEN LADEN"
1990 PRINT
2000 PRINT
2010 INPUT"DATEINAME";DN$
2020 PRINT
2030 OPEN 2,8,2,DN$+",S,R"
2040 K=0
2050 IF DS<>0 THEN PRINT DS$:GOTO 2110
2060 IF K=MA THEN PRINT"DATEI VOLL":GOTO 2110
2070 K=K+1
2080 PRINTK;"<1xCRSR UP>"
```

```
2090 INPUT#2,N$(K)
2100 IF ST=0 THEN 2060
2110 DCLOSE#2
2120 PRINT
2130 PRINT
2140 PRINT"          ZURUECK INS MENUE"
2150 PRINT"       BELIEBIGE TASTE DRUECKEN"
2160 GETKEY M$
2170 RETURN
2180 :
2190 :
2200 :
2210 REM ADRESSENDATEI ABSPEICHERN
2220 GOSUB 3550
2230 PRINT"<CLR>"
2240 PRINT
2250 PRINT
2260 PRINT"       ADRESSEN ABSPEICHERN"
2270 PRINT
2280 PRINT
2290 IF K=0 THEN PRINT"DATEI LEER":PRINT:GOTO 2500
2300 INPUT"DATEINAME";DN$
2310 PRINT
2320 OPEN 2,8,2,DN$+",S,W"
2330 F=DS
2340 IF F=0 THEN 2450
2350 PRINT DS$
2360 CLOSE 2
2370 IF F<>63 THEN 2500
2380 PRINT
2390 PRINT"DATEI UEBERSCHREIBEN ? (J/N)"
2400 PRINT
2410 GETKEY M$
2420 IF M$="J" THEN DN$="@:"+DN$:GOTO 2320
2430 IF M$="N" THEN 2500
2440 GOTO 2410
2450 FOR I=1 TO K
2460 PRINT I;"<1xCRSR UP>"
2470 PRINT#2,N$(I)
2480 NEXT I
2490 DCLOSE#2
2500 PRINT
```

```
2510 PRINT
2520 PRINT"         ZURUECK INS MENUE"
2530 PRINT"      BELIEBIGE TASTE DRUECKEN"
2540 GETKEY M$
2550 RETURN
2560 :
2570 :
2580 :
2590 REM ADRESSENDATEI AUSDRUCKEN
2600 GOSUB 3370
2610 PRINT"<CLR>"
2620 PRINT
2630 PRINT
2640 PRINT"      ADRESSEN AUSDRUCKEN"
2650 PRINT
2660 PRINT
2670 OPEN4,4
2680 FOR I=1 TO K
2690 PRINT#4,LEFT$(N$(I),LN);
2700 PRINT#4," ";
2710 PRINT#4,MID$(N$(I),LN+1,LV);
2720 PRINT#4," ";
2730 PRINT#4,MID$(N$(I),LN+LV+1,LS);
2740 PRINT#4," ";
2750 PRINT#4,MID$(N$(I),LN+LV+LS+1,LO);
2760 PRINT#4," ";
2770 PRINT#4,RIGHT$(N$(I),1)
2780 IF I-INT(I/64)*64=0 THEN BEGIN
2790 FOR J=1 TO 8
2800 PRINT#4
2810 NEXT J
2820 BEND
2830 NEXT I
2840 CLOSE 4
2850 PRINT
2860 PRINT
2870 PRINT"         ZURUECK INS MENUE"
2880 PRINT"      BELIEBIGE TASTE DRUECKEN"
2890 GETKEY M$
2900 RETURN
2910 :
2920 :
```

```
2930 :
2940 REM ADRESSEN AUF ETIKETTEN DRUCKEN
2950 GOSUB 3370
2960 PRINT"<CLR>"
2970 PRINT
2980 PRINT
2990 PRINT" ADRESSEN AUF ETIKETTEN AUSDRUCKEN"
3000 PRINT
3010 PRINT
3020 OPEN4,4
3030 FOR I=1 TO K
3040 PRINT#4
3050 PRINT#4
3060 A=VAL(RIGHT$(N$(I),1))
3070 PRINT#4,AR$(A)
3080 PRINT#4,LEFT$(N$(I),LN+LV)
3090 PRINT#4,MID$(N$(I),LN+LV+1,LS)
3100 PRINT#4
3110 PRINT#4,MID$(N$(I),LN+LV+LS+1,LO)
3120 PRINT#4
3130 PRINT#4
3140 NEXT I
3150 CLOSE 4
3160 PRINT
3170 PRINT
3180 PRINT"          ZURUECK INS MENUE"
3190 PRINT"       BELIEBIGE TASTE DRUECKEN"
3200 GETKEY M$
3210 RETURN
3220 :
3230 :
3240 :
3250 REM DATENSATZ LOESCHEN
3260 FOR J=I+1 TO K
3270 N$(J-1)=N$(J)
3280 NEXT J
3290 K=K-1
3300 PRINT"<8xCRSR DOWN>"
3310 PRINT"GELOESCHT"
3320 PRINT"<2xCRSR UP>"
3330 RETURN
3340 :
```

```
3350 :
3360 :
3370 REM ABFRAGE OB DRUCKER EINGESCHALTET
3380 TRAP 3430
3390 OPEN4,4,0," "
3400 CLOSE 4
3410 RETURN
3420 :
3430 IF ER<>5 THEN RESUME 3410
3440 CLOSE 4
3450 PRINT"<CLR>"
3460 PRINT
3470 PRINT
3480 PRINT "      BITTE DRUCKER EINSCHALTEN"
3490 PRINT "     UND BELIEBIGE TASTE DRUECKEN"
3500 GETKEY M$
3510 RESUME
3520 :
3530 :
3540 :
3550 REM ABFRAGE OB FLOPPY EINGESCHALTET
3560 TRAP 3610
3570 OPEN4,8,15,"I"
3580 CLOSE4
3590 RETURN
3600 :
3610 CLOSE4
3620 IF ER<>5 THEN RESUME 3590
3630 PRINT"<CLR>"
3640 PRINT
3650 PRINT
3660 PRINT "       BITTE FLOPPY EINSCHALTEN"
3670 PRINT "     UND BELIEBIGE TASTE DRUECKEN"
3680 GETKEY M$
3690 RESUME
```

Wir wollen uns nun einige wichtige Einzelheiten zum Programmaufbau ansehen. Über die Größe der Datensätze geben die ersten Zeilen Auskunft, so daß Sie sie leicht Ihren eigenen Wünschen anpassen können.

Zunächst gibt die Variable MA die maximale Anzahl der Datensätze an. Sie ist hier mit 200 vorbelegt, kann aber noch um einiges erhöht werden, falls

dies nötig sein sollte. Die einzige Begrenzung ist hierbei in der Größe des Variablenspeichers gegeben.

Die Variable K gibt an, wieviele Datensätze sich gerade im Speicher befinden. Sie kann an jeder Stelle des Programms abgefragt werden, sofern Sie noch zusätzliche Kontrollmöglichkeiten einbauen. Die folgenden Variablen LN, LV, LS und LO geben die Längen für die einzelnen Datenwerte an und können bei Bedarf ebenfalls verändert werden.

Sämtliche Datenwerte eines Datensatzes bilden zusammen einen String, der genau 72 Zeichen lang und ein Element des Feldes N$(I) ist. Zeile 130 führt die Dimensionierung für dieses eindimensionale Feld mit MA Elementen durch.

In jedem Datensatz haben die einzelnen Werte ihren festen Platz, wie die folgende Aufstellung zeigt:

Name	15 Zeichen von	Position 1 bis 15
Vorname	15 Zeichen von	Position 16 bis 29
Straße	20 Zeichen von	Position 31 bis 50
PLZ/Ort	21 Zeichen von	Position 51 bis 71
Anredecode	1 Zeichen	Position 72

Die einzelnen Werte können mit Hilfe der Stringoperationen leicht aus dem Gesamtstring eliminiert bzw. wieder zu diesem zusammengesetzt werden. Werden bei der Eingabe die zulässigen Längen überschritten, werden alle zusätzlichen Zeichen abgeschnitten. Werden dagegen weniger Zeichen eingegeben, werden sie mit Leerzeichen aufgefüllt, so daß ihre vorgeschriebene Länge in jedem Fall konstant bleibt.

Die einzelnen Anreden werden ebenfalls in einem Feld AR$(I) entsprechend ihrem Code abgelegt. Wirklich ausgegeben werden sie aber nur beim Etikettenausdruck.

Das Menü ist ähnlich aufgebaut wie dasjenige, das wir bereits im Programm zur Volumenberechnung verschiedener Körper kennengelernt haben. Die einzelnen Menüpunkte werden mit Hilfe der GETKEY-Anweisung in Zeile 490 als Strings gelesen und dann in eine numerische Variable umgeformt. Eine Ausnahme bildet lediglich der Punkt E, der das Programm beendet. Schließlich ruft die ON...GOSUB-Anweiung in Zeile 530 die entsprechenden Routinen auf.

Das Unterprogramm ab Zeile 580 dient zur Eingabe und Aufbereitung der Eingabewerte, die gleichzeitig ihre vorgeschriebene Länge erhalten (s.o.). In Zeile 840 werden sie schließlich zum Gesamtstring N$(K) zusammengesetzt.

Etwas komplizierter dagegen ist das Teilprogramm ab Zeile 980, das die Adressen löscht und ändert. Zunächst wird der gesuchte Name angefordert und sämtliche Datensätze nach ihm durchsucht. Wird der Datensatz mit dem betreffenden Namen gefunden, erhält die Variable I die entsprechende Datensatznummer, wird er dagegen nicht gefunden, erscheint der Hinweis "NICHT GEFUNDEN" auf dem Bildschirm.

Daraufhin werden sämtliche Werte eines Datensatzes an derselben Stelle wie bei ihrer Eingabe unter Menüpunkt 1 auf dem Bildschirm ausgegeben. Zeile 1310 setzt den Cursor wieder zurück in die oberste Eingabezeile (Anredecode), wobei dann jeder Wert, der bereits mit PRINT ausgegeben wurde, mit einer INPUT-Anweisung "überlagert" wird. Dabei muß der Cursor jeweils um 12 Zeichen nach rechts bewegt werden, ohne die darunterliegenden Angaben zu überschreiben. Dank dieses Tricks ist es möglich, daß Ihnen bei INPUT der Eingabewert bereits vorpräsentiert wird, so daß Sie ihn nur, falls erforderlich, korrigieren müssen, um ihn dann durch Drücken der RETURN-Taste wieder ins Programm zu übernehmen.

Zum Löschen eines Datensatzes wird der Anredecode außer auf die sonst zulässigen Werte von 0 bis 5 auch auf den Buchstaben L abgefragt (Zeile 1340). Daraufhin wird das Unterprogramm in Zeile 3250 aufgerufen, das den Löschvorgang vornimmt und alle Datensätze mit höherer Nummer um eine Stelle nach unten aufrückt. Die derzeitige Gesamtanzahl der Datensätze im Speicher wird dabei um eins reduziert (Variable K).

Alphabetisch sortiert werden die Datensätze im Unterprogramm ab Zeile 1620, wobei das schnelle Shell-Sortierverfahren zur Anwendung kommt, das wir bereits kennengelernt haben.

In den nächsten beiden Unterprogrammen werden die Datensätze von Diskette gelesen bzw. auf sie geschrieben. Allerdings hätten wir statt der OPEN-Anweisung auch DOPEN verwenden können. Da in unserem Beispiel der Dateiname nicht als Konstante, sondern als Variable DN$ vorgegeben ist, funktioniert die Anweisung aber nur, wenn wir DN$ in Klammern setzen, wie z.B.

```
DOPEN#2, (DN$)
```

Da beim Einlesen die Anzahl der Datensätze zunächst noch unbekannt ist, fragen wir mit Hilfe der Statusvariablen ST das Dateiende ab. In diesem Fall nimmt sie den Wert 64 an, während sie ansonsten Null ist, sofern kein anderer Fehler auftritt.

Die Fehlerabfrage des Disketten-Betriebssystems erfolgt über die reservierte Variable DS bzw. DS$. Tritt ein Fehler auf, wird die Floppy-Operation abgebrochen. Eine Ausnahme bildet beim Schreiben der Fehlercode 63 (FILE EXISTS), der auftritt, wenn eine gleichnamige Datei bereits auf der Diskette vorhanden ist. Hier wird der Benutzer zunächst gefragt, ob er ein Überschreiben dieser Datei wünscht. Ist dies der Fall, wird dem Dateinamen ein Klammeraffe mit Doppelpunkt vorangestellt, damit die gleichnamige Datei überschrieben werden kann.

Vor dem Einsprung in die Floppy-Routinen wird zunächst das Unterprogramm in Zeile 3550 aufgerufen, das mit Hilfe der TRAP-Anweisung feststellt, ob die Floppy eingeschaltet ist. Ist dies nicht der Fall, wird der Bediener aufgefordert, es zu tun. Eine ähnliche Routine steht ab Zeile 3370 für den Drucker.

Zu den Druckerroutinen selbst (Zeile 2590 bzw. 2940) gibt es nicht mehr viel zu sagen. Lediglich beim Listenausdruck findet ab Zeile 2780 eine Prüfung statt, ob bereits 64 Zeilen hintereinander ausgedruckt wurden. Ist dies der Fall, wird durch 8 Leerzeilen ein Seitenvorschub erzeugt.

Hier noch ein wichtiger Hinweis für den Fall, daß Sie das Programm für Ihre eigenen Belange anpassen möchten. Die Länge des Gesamtstrings, der auf Diskette abgespeichert wird, darf auf keinen Fall 160 Zeichen überschreiten, da der BASIC-Eingabepuffer ansonsten überläuft. Wenn Sie längere Datensätze verarbeiten möchten, empfiehlt es sich, sie in mehrere Strings zu zerlegen und diese hintereinander abzuspeichern (z.B. N1$(I) und N2$(I)). Beachten Sie aber, daß kein String im Speicher länger als 255 Zeichen sein darf.

6.5 Relative Dateiverwaltung auf Diskette

Die Dateien, die wir bisher kennengelernt haben, befanden sich ausschließlich im Arbeitsspeicher des Computers und konnten nur komplett bearbeitet werden. Ebenso wurden sie als Ganzes auf Diskette gespeichert und wieder geladen. Nehmen wir einmal das sequentielle Adressenverwaltungsprogramm als Beispiel, mit dem wir nur so viele Adressen erfassen und verarbeiten können, wie der Speicher des Computers aufnehmen kann.

Wollen wir auch nur eine einzige Adresse ändern, müssen wir zuvor sämtliche Datensätze der sequentiellen Datei einlesen und nach der Bearbeitung wieder neu auf Diskette schreiben. Sie können sich sicher vorstellen, daß dies eine ziemlich aufwendige Angelegenheit ist.

Diese Einschränkung ist weitestgehend durch die relative Dateiverwaltung zu umgehen. Dabei sind die einzelnen Datensätze nicht im Speicher des Computers, sondern in sogenannten Records auf der Diskette abgelegt, wobei jedem einzelnen Record ein Datensatz zugeordnet ist.

Record ist ein amerikanisches Wort und heißt übersetzt ebenfalls Datensatz. Zur besseren Unterscheidung treffen wir hier deshalb die Vereinbarung, daß wir unter "Record" den auf der Diskette reservierten Speicherplatz für einen Eintrag verstehen, während ein "Datensatz" weiterhin das bleibt, was er bisher war, nämlich ein Eintrag in der Datei, der mehrere Datenposten oder Werte umfassen kann. In unserer Adressenverwaltung z.B. enthält ein Datensatz Anredecode, Nachname, Vorname, Straße und PLZ/Ort.

Beim Einrichten einer relativen Datei wird eine bestimmte Anzahl von Records mit fester Länge reserviert, die bis zu 254 Zeichen umfassen können. Diese Records belegen, ähnlich wie bei einer sequentiellen Datei mit konstanter Datensatzlänge, fortlaufende, durch interne Zeiger verknüpfte Sektoren, wobei das DOS aufgrund einer intern angelegten Tabelle (Sidesektoren) schnell auf jeden einzelnen Record zugreifen kann. Dazu ist lediglich die fortlaufende Nummer des Records anzugeben.

Bevor wir jedoch mit einer relativen Datei arbeiten können, müssen wir sie anlegen. Betrachten wir dazu folgendes Beispiel:

```
10 DOPEN#1, "REL-DATEI", L30
20 GOSUB 1000
30 RECORD#1, 50
40 GOSUB 1000
50 PRINT#1, CHR$(255)
60 GOSUB 1000
70 DCLOSE#1
80 END
1000 IF DS=0 OR DS=50 THEN RETURN
1010 PRINT DS$
1020 END
```

Zunächst öffnet die DOPEN-Anweisung in Zeile 10 das File 1 für die Datei "REL-DATEI". Dies geschieht genauso wie das Öffnen einer sequentiellen Datei, jedoch mit dem zusätzlichen Parameter L30. Dieser Parameter gibt an, daß die Datei Records von 30 Zeichen Länge erhalten soll. Damit ist aber noch kein Speicherplatz für die Records reserviert und noch nicht bestimmt, wieviele Records die Datei insgesamt erhalten soll. Im Anschluß an die DOPEN-Anweisung wird das Unterprogramm in Zeile 1000 aufgerufen, welches prüft, ob das DOS eine Fehlermeldung erzeugt hat.

Zeile 30 enthält nun die RECORD-Anweisung, die sich auf das in der DOPEN-Anweisung angegebene File 1 bezieht und den internen Record-Zeiger auf den 50. Record setzt. Da dieser noch gar nicht existiert, erscheint zunächst ein DOS-Fehler mit der Nummer 50 (RECORD NOT PRESENT), den wir aber ruhigen Gewissens ignorieren können. Deshalb ist das Unterprogramm ab Zeile 1000 so aufgebaut, daß es beim Auftreten dieses Fehlers sofort ins Hauptprogramm zurückspringt.

Beschrieben und wirklich angelegt wird der 50. Record erst in Zeile 50. Die Schreibanweisung PRINT#1,CHR$(255) belegt ihn nämlich mit dem Kontrollcode CHR$(255). Bei diesem Schreibvorgang stellt das DOS fest, daß die vorangehenden 49 Records noch nicht existieren und legt sie gleichzeitig mit an, indem es ebenfalls den Code CHR$(255) in sie einträgt. Zeile 70 schließt das File, womit das Anlegen der relativen Datei beendet ist.

Soll eine relative Datei zu einem späteren Zeitpunkt erweitert werden, sind die gleichen Anweisungen nochmals auszuführen, wobei jedoch in der RECORD-Anweisung die gewünschte Anzahl der Records angegeben sein muß. Dadurch werden die bereits vorher angelegten und beschriebenen Records nicht berührt. Die Recordlänge in der DOPEN-Anweisung muß mit derjenigen in der bereits bestehenden Datei übereinstimmen und kann nachträglich nicht mehr geändert werden. Schließlich ist noch zu beachten, daß eine relative Datei maximal 720 Records enthalten darf.

Die folgenden beiden Beispiele zeigen, wie wir auf Records einer bereits bestehenden Datei zugreifen können, um Daten zu schreiben oder zu lesen. Dabei arbeiten wir wieder mit der relativen Datei "REL-DATEI", die wir soeben angelegt haben:

```
5 REM RECORD SCHREIBEN
10 DOPEN#1,"REL-DATEI"
20 GOSUB 1000
30 INPUT "WELCHER RECORD"; R
40 RECORD#1, R, 1
50 GOSUB 1000
60 PRINT#1, "DIES IST DER"; R;". RECORD"
70 GOSUB 1000
80 DCLOSE#1
90 END
1000 IF DS=0 OR DS=50 THEN RETURN
1010 PRINT DS$
1020 END
```

Zeile 10 eröffnet wieder die relative Datei "REL-DATEI". Diesmal brauchen wir jedoch keinen Parameter, der die Länge der einzelnen Records angibt, denn wir haben die Datei ja bereits angelegt und wollen lediglich auf einzelne Records zugreifen.

In Zeile 30 geben Sie jetzt die Nummer des Records ein, den Sie beschreiben möchten. Daraufhin setzt Zeile 40 mit der RECORD-Anweisung den internen Zeiger auf den betreffenden Record. Der letzte Parameter, die Ziffer 1, gibt an, daß der Record ab der ersten Stelle beschrieben werden soll. Wir könnten auch jede beliebige andere Position innerhalb der Recordlänge angeben, müssen aber darauf achten, daß der Datensatz in den Record noch hineinpaßt. Für die Länge des Records sollte immer ein Zeichen mehr reserviert werden, als der Datensatz umfaßt, da dieser beim Schreiben jedesmal mit einem Carriage Return (CHR$(13)) abgeschlossen wird.

Nachdem nun der Record und die Position in ihm festgelegt sind, können wir Daten in ihn hineinschreiben. Zu Übungszwecken schreiben wir jeweils eine Kennzeichnung hinein, damit wir später feststellen können, ob unser Programm richtig arbeitet. Wählen wir z.B. den 10. Record aus, so erhält dieser den Datensatz

DIES IST DER 10. RECORD

Einschließlich dem Carriage Return liegt die Länge des Datensatzes mit 24 Zeichen noch innerhalb der Recordlänge von 30 Zeichen.

Es besteht grundsätzlich die Möglichkeit, in einem Record mehrere Eintragungen vorzunehmen, z.B. Artikelnummer (6 Zeichen), Artikelbezeichnung (20 Zeichen) und Preis (8 Zeichen). Zur Bestimmung der erforderlichen Recordlänge ist allerdings zu jedem einzelnen Datenwert noch ein Zeichen hinzuzuzählen, da auch hier ein Carriage Return jeweils mit abgespeichert wird. In diesem Fall müßte also die Recordlänge mindestens 6+1+20+1+8+1 = 37 Zeichen umfassen, wobei die Artikelnummer an der 1., die Artikelbeschreibung an der 8. und der Preis an der 29. Stelle im Record beginnt. Für den Eintrag jeder dieser Werte muß die RECORD-Anweisung mit der entsprechenden Positionierung erneut ausgeführt werden, bevor die Werte mit PRINT# geschrieben werden können. Auch darf in einem solchen Fall der Record nur von vorne nach hinten beschrieben werden, da anderenfalls die hinteren Einträge zerstört werden.

In unserem Fall heißt dies, daß wir zuerst die Artikelnummer, dann die Artikelbeschreibung und schließlich den Preis eintragen müssen. Würden wir nämlich die Artikelnummer zuletzt eintragen, würden Artikelbezeichnung und Preis gelöscht.

Aber nun zurück zu unserem Beispiel. Wir wollen jetzt einmal sehen, ob der 10. Record (oder ein anderer) den richtigen Eintrag erhalten hat. Dazu schreiben wir ein kleines Programm, das die Datensätze wieder einliest und auf dem Bildschirm ausgibt:

```
5 REM RECORD LESEN
10 DOPEN#1,"REL-DATEI"
20 GOSUB 1000
30 INPUT "WELCHER RECORD"; R
40 RECORD#1, R, 1
50 GOSUB 1000
60 INPUT#1, A$
70 GOSUB 1000
80 PRINT A$
90 DCLOSE#1
100 END
1000 IF DS=0 OR DS=50 THEN RETURN
1010 PRINT D$
1020 END
```

Dieses Programm ist ähnlich aufgebaut, wie dasjenige zum Schreiben von Records. Statt einer PRINT#- enthält es diesmal eine INPUT#-Anweisung, die den Record liest und der Variablen A$ zuordnet, deren Inhalt dann auf dem Bildschirm erscheint. Haben wir nun, wie oben erklärt, den 10. Record korrekt geschrieben, muß er nach Ausführung des Programms wieder gelesen werden können und auf dem Bildschirm erscheinen:

```
DIES IST DER 10. RECORD

READY.
```

Selbstverständlich kann auch beim Lesen an jeder beliebigen Stelle im Record begonnen werden; dabei ist, wie beim Schreibvorgang, die entsprechende Position in der RECORD-Anweisung anzugeben. In diesem Fall wird der Inhalt bis zum nächsten Carriage-Return ausgelesen.

6.5.1 Adressenverwaltungsprogramm (relativ)

Das im letzten Abschnitt besprochene sequentielle Adressenverwaltungsprogramm wurde nachfolgend für die relative Dateiverwaltung umgeschrieben. Dabei befinden sich die Datensätze nicht mehr im Speicher, sondern in einer relativen Datei auf Diskette.

Jedesmal, wenn ein Datensatz bearbeitet werden soll, wird er entweder neu eingegeben oder einzeln aus der relativen Datei gelesen und nach der Bearbeitung wieder auf Diskette geschrieben. Doch woher weiß nun das Programm, in welchem Record sich welche Adresse befindet?

Wird die erste eingegebene Adresse in Record Nr. 1, die zweite in Record Nr. 2 usw. abgelegt, ist die Sache noch relativ einfach, denn wir müssen dann nur die betreffende Recordnummer aufrufen. Schwieriger wird es, wenn wir die Adressen alphabetisch sortiert oder einzelne Adressen aus der Datei entfernt haben.

Des Rätsels Lösung heißt: indexsequentielle Dateiverwaltung. Indexsequentiell bedeutet, daß von jedem Record die Recordnummer und ein spezieller Index zusätzlich in einer sequentiellen Datei abgelegt wird, die zusammen mit der relativen Datei auf Diskette geschrieben wird. Dieser Index dient dann als Suchkriterium zum Auffinden eines Datensatzes.

Für unser Adressenverwaltungsprogramm wurde der Name als Index gewählt. Bevor nun Datensätze angelegt oder bearbeitet werden können, wird zunächst die Indexdatei in den Arbeistspeicher des Computers geladen. Sie bleibt dort solange, wie wir mit der relativen Datei arbeiten und wird im Anschluß daran wieder auf Diskette geschrieben.

Wenn wir nun einen Datensatz anlegen, "weiß" die Indexdatei, welche Records bereits belegt und welche noch frei sind. Sie sucht den nächstverfügbaren Record heraus und legt dort den Datensatz ab. Gleichzeitig erhält die Indexdatei einen neuen Eintrag, der den Nachnamen als Index und die dazugehörige Recordnummer enthält.

Suchen wir nun umgekehrt den Datensatz zum Namen Müller, wird zunächst die Indexdatei durchsucht, um festzustellen, ob der Datensatz in der Datei enthalten ist. Ist dies der Fall, steht auch gleichzeitig die Recordnummer zur Verfügung, die dann leicht aufgerufen werden kann.

Bei der sequentiellen Dateiverwaltung benötigten wir keine zusätzliche Indexdatei, da ja sämtliche Datensätze zu jeder Zeit vollständig im Speicher abgelegt waren und wir sie nur nach dem richtigen Namen durchsuchen mußten. Natürlich wäre es auch möglich, analog zur sequentiellen Dateiverwaltung sämtliche Records von Diskette einzulesen, um einen bestimmten Namen ausfindig zu machen. Dies ist jedoch eine sehr mühsame und zeitraubende Angelegenheit, da das Einlesen von Diskette viel mehr Zeit in Anspruch nimmt, als ein Feld im Arbeitsspeicher zu durchsuchen.

Die Indexdatei benötigt außerdem viel weniger Speicherplatz als die gesamte Datei, was zusätzlich Speicherplatzproblemen entgegenwirkt. Wenn wir nun unsere Adressendatei alphabetisch sortieren, geschieht dies letztlich nur mit der Indexdatei, wobei die Datensätze selbst im gleichen Record gespeichert bleiben. Ähnliches geschieht auch beim Löschen eines Datensatzes. Dabei wird nicht der betreffende Record, sondern lediglich der Eintrag in der Indexdatei gelöscht bzw. für neue Einträge freigegeben.

Bevor wir uns detailliert mit unserem Programm befassen, werfen wir zunächst einen Blick auf das Menü, das auch hier nach dem Starten des Programms auf dem Bildschirm erscheint:

```
            ADRESSENVERWALTUNG
            ==================

ADRESSEN SCHREIBEN/FORTSCHREIBEN        1

ADRESSEN AENDERN/LOESCHEN               2

ADRESSEN SORTIEREN                      3

RELATIVE DATEI OEFFNEN                  4

RELATIVE DATEI ANLEGEN                  5

ADRESSEN AUSDRUCKEN                     6

ADRESSEN AUF ETIKETTEN AUSDRUCKEN       7

E N D E                                 E


BITTE WAEHLEN SIE
```

Vielleicht ist Ihnen aufgefallen, daß die meisten Menüpunkte mit denen im sequentiellen Adressenverwaltungsprogramm identisch sind, und sich lediglich die Punkte 4 und 5 unterscheiden.

Wir wissen bereits, daß wir eine relative Datei anlegen müssen, bevor wir darin Eintragungen vornehmen können. Genau dies geschieht durch den Menüpunkt 5. Zusätzlich wird hier noch die Indexdatei erzeugt und auf Diskette geschrieben. Wenn Sie sich jetzt das Disketten-Inhaltsverzeichnis ansehen, enthält es eine relative und eine sequentielle Datei, die beide neu angelegt sind, aber noch keine Einträge enthalten. Damit es keine Verwechslungen mit den Dateinamen gibt, erhält die relative Datei den von Ihnen gewählten Namen und die sequentielle Indexdatei den gleichen Namen mit einem angehängten "-I". Wenn Sie nun z.B. eine Datei unter dem Namen "KUNDEN" anlegen, erhält die relative Datei den Namen "KUNDEN", während er für die Indexdatei "KUNDEN-I" lautet.

Bevor Sie Datensätze anlegen oder bearbeiten können, müssen Sie auf jeden Fall Menüpunkt 4 anwählen, der die relative Datei öffnet. Öffnen ist hier nicht nur unbedingt im Sinne der OPEN- bzw. DOPEN-Anweisung zu verstehen, sondern bedeutet vielmehr eine Initialisierung der Adressendatei. Dabei wird die Indexdatei in den Speicher geladen und gleichzeitig festgestellt, welche Datensätze belegt und welche noch frei sind.

Erst jetzt können Sie daran gehen, Adressen einzugeben oder sie anderweitig zu bearbeiten. Die restlichen Menüpunkte haben für den Bediener die gleiche Funktion wie bei der sequentiellen Adressenverwaltung. Nur bei Beendigung des Programms muß hier unbedingt die E-Taste gedrückt werden, damit die Indexdatei auf Diskette geschrieben werden kann. Wenn Sie dies vergessen und den Computer einfach ausschalten oder das Programm beispielsweise mit der STOP-Taste oder dem Reset-Schalter abbrechen, kann es unter Umständen zu erheblichen Fehlern beim späteren Wiedereinlesen der Datensätze kommen.

Nachfolgend das Programmlisting:

```
10 REM ADRESSENVERWALTUNGSPROGRAMM (RELATIV)
20 REM=====================================
30 REM
40 REM
50 MA=200:REM MAX.ANZAHL DER DATENSAETZE
60 K=0:REM DATENSAETZE IM SPEICHER
70 LN=15:REM ZEICHEN LAENGE NAMEN
80 LV=15:REM ZEICHEN LAENGE VORNAMEN
90 LS=20:REM ZEICHEN LAENGE STRASSE
100 LO=21:REM ZEICHEN LAENGE PLZ/ORT
110 LA=1:REM ZEICHEN LAENGE ANREDECODE
120 FL=0:REM FLAG OB DATEI IM SPEICHER
130 L=LN+LV+LS+LO+LA+1
140 G$="**************************"
150 N$="                          "
160 DIM N$(MA)
170 DIM AR$(5)
180 AR$(0)=""
190 AR$(1)="HERRN"
200 AR$(2)="FRAU"
210 AR$(3)="FRAEULEIN"
220 AR$(4)="HERRN UND FRAU"
230 AR$(5)="FIRMA"
240 :
250 :
260 :
270 REM MENUE
280 PRINT"<CLR>"
290 PRINT TAB(10);"ADRESSENVERWALTUNG"
300 PRINT TAB(10);"=================="
310 PRINT
320 PRINT
330 PRINT"ADRESSEN SCHREIBEN/FORTSCHREIBEN          1"
340 PRINT
350 PRINT"ADRESSEN AENDERN/LOESCHEN                 2"
360 PRINT
370 PRINT"ADRESSEN SORTIEREN                        3"
380 PRINT
390 PRINT"RELATIVE DATEI OEFFNEN                    4"
400 PRINT
410 PRINT"RELATIVE DATEI ANLEGEN                    5"
420 PRINT
```

```
430 PRINT"ADRESSEN AUSDRUCKEN                    6"
440 PRINT
450 PRINT"ADRESSEN AUF ETIKETTEN AUSDRUCKEN      7"
460 PRINT
470 PRINT
480 PRINT"E N D E                                E"
490 PRINT
500 PRINT
510 PRINT"BITTE WAEHLEN SIE"
520 GETKEY M$
530 IF M$="E" THEN 4330
540 M=VAL(M$)
550 IF M<1 OR M>7 THEN 520
560 ON M GOSUB 610,1130,1940,2250,2560,3020,3440
570 GOTO 270
580 :
590 :
600 :
610 REM ADRESSEN SCHREIBEN/FORTSCHREIBEN
620 GOSUB 4150
630 PRINT"<CLR>"
640 PRINT
650 PRINT
660 PRINT"    ADRESSEN SCHREIBEN/FORTSCHREIBEN    "
670 PRINT
680 PRINT
690 IF DN$="" THEN PRINT"BITTE DATEI OEFFNEN":GOTO 1030
700 OPEN5,8,5,DN$:CLOSE5
710 IF DS<>0 THEN PRINT DS$:GOTO1030
720 K=K+1
730 PRINT K
740 PRINT
750 IF K>MA THEN PRINT"DATEI VOLL":GOTO1000
760 PRINT"ANREDE (0-5)";:INPUT A$
770 A=VAL(A$)
780 IF A<0 OR A>5 OR LEN(A$)<>1 THEN PRINT"<2xCRSR UP>"
                                                    :GOTO760
790 PRINT
800 PRINT"NAME        ";:INPUTNA$
810 PRINT
820 NA$=LEFT$(NA$+N$,LN)
830 PRINT"VORNAME     ";:INPUTNV$
```

```
840 PRINT
850 NV$=LEFT$(NV$+N$,LV)
860 PRINT"STRASSE      ";:INPUTNS$
870 PRINT
880 NS$=LEFT$(NS$+N$,LS)
890 PRINT"PLZ, ORT     ";:INPUTNO$
900 NO$=LEFT$(NO$+N$,LO)
910 R$=NA$+NV$+NS$+NO$+RIGHT$(STR$(A),1)
920 OPEN2,8,2,DN$
930 D=VAL(MID$(N$(K),LN+1,10))
950 RECORD#2,D,1
960 PRINT#2,R$
970 DCLOSE#2
980 FL=1
990 N$(K)=NA$+STR$(D)
1000 PRINT
1010 PRINT
1020 PRINT"NAECHSTE EINGABE - RETURN-TASTE DRUECKEN"
1030 PRINT
1040 PRINT"ZURUECK INS MENUE - LEERTASTE DRUECKEN"
1050 GETKEY M$
1060 IF M$=CHR$(13) AND DN$<>"" THEN 610
1070 IF M$=" " THEN CLOSE2:GOTO 1090
1080 GOTO 1050
1090 RETURN
1100 :
1110 :
1120 :
1130 REM ADRESSEN AENDERN/LOESCHEN
1140 GOSUB 4150
1150 PRINT"<CLR>"
1160 PRINT
1170 PRINT
1180 PRINT"        ADRESSEN AENDERN/LOESCHEN"
1190 PRINT
1200 PRINT
1210 IF DN$="" THEN PRINT"BITTE DATEI OEFFNEN":GOTO1840
1220 OPEN5,8,2,DN$:CLOSE5
1230 IF DS<>0 THEN PRINT DS$:GOTO1840
1240 PRINT"NAME        ";:INPUT NA$
1250 NA$=LEFT$(NA$+N$,LN)
1260 I=1
```

```
1270 DO UNTIL I>K
1280 IF LEFT$(N$(I),LN)=NA$ THEN 1340
1290 I=I+1
1300 LOOP
1310 PRINT
1320 PRINT"NICHT GEFUNDEN"
1330 GOTO1810
1340 OPEN2,8,2,DN$
1350 D=VAL(MID$(N$(I),LN+1,10))
1370 RECORD#2,D,1
1380 INPUT#2,R$
1390 DCLOSE#2
1400 NA$=LEFT$(R$,LN)
1410 NV$=MID$(R$,LN+1,LV)
1420 NS$=MID$(R$,LN+LV+1,LS)
1430 NO$=MID$(R$,LN+LV+LS+1,LO)
1440 A$=RIGHT$(R$,1)
1450 PRINTI
1460 PRINT
1470 PRINT"ANREDE (0-5)  ";A$
1480 PRINT
1490 PRINT"NAME          ";NA$
1500 PRINT
1510 PRINT"VORNAME       ";NV$
1520 PRINT
1530 PRINT"STRASSE       ";NS$
1540 PRINT
1550 PRINT"PLZ, ORT      ";NO$
1560 PRINT"<10xCRSR UP>"
1570 PRINT"<12xCRSR RIGHT>";:INPUT A$
1580 A=VAL(A$)
1590 IF A$="L" THEN GOSUB 3820:GOTO 1810
1600 IF A<0 OR A>5 OR LEN(A$)<>1 THEN PRINT"<2xCRSR UP>"
                                                :GOTO 1570
1610 PRINT
1620 PRINT"<12xCRSR RIGHT>";:INPUT NA$
1630 PRINT
1640 NA$=LEFT$(NA$+N$,LN)
1650 PRINT"<12xCRSR RIGHT>";:INPUT NV$
1660 PRINT
1670 NV$=LEFT$(NV$+N$,LV)
1680 PRINT"<12xCRSR RIGHT>";:INPUT NS$
```

```
1690 PRINT
1700 NS$=LEFT$(NS$+N$,LS)
1710 PRINT"<12xCRSR RIGHT>";:INPUT NO$
1720 NO$=LEFT$(NO$+N$,LO)
1730 R$=NA$+NV$+NS$+NO$+RIGHT$(STR$(A),1)
1740 OPEN 2,8,2,DN$
1750 D=VAL(MID$(N$(I),LN+1,10))
1770 RECORD#2,D,1
1780 PRINT#2,R$
1790 DCLOSE#2
1800 N$(I)=NA$+STR$(D)
1810 PRINT
1820 PRINT
1830 PRINT"NAECHSTE EINGABE - RETURN-TASTE DRUECKEN"
1840 PRINT
1850 PRINT"ZURUECK INS MENUE - LEERTASTE DRUECKEN"
1860 GETKEY M$
1870 IF M$=CHR$(13) AND DN$<>"" THEN 1130
1880 IF M$=" " THEN 1900
1890 GOTO 1860
1900 RETURN
1910 :
1920 :
1930 :
1940 REM ADRESSEN SORTIEREN
1950 PRINT"<CLR>"
1960 PRINT
1970 PRINT
1980 PRINT"          ADRESSEN SORTIEREN        "
1990 PRINT
2000 PRINT
2010 G=INT(K/2)
2020 DO WHILE G>0
2030 FOR I=G+1 TO K
2040 J=I-G
2050 DO WHILE J>0
2060 IF N$(J)>N$(J+G) THEN BEGIN
2070 A$=N$(J+G)
2080 N$(J+G)=N$(J)
2090 N$(J)=A$
2100 J=J-G
2110 BEND:ELSE J=0
```

```
2120 LOOP
2130 NEXT
2140 G=INT(G/2)
2150 LOOP
2160 PRINT
2170 PRINT
2180 PRINT"          ZURUECK INS MENUE"
2190 PRINT"       BELIEBIGE TASTE DRUECKEN"
2200 GETKEY M$
2210 RETURN
2220 :
2230 :
2240 :
2250 REM RELATIVE DATEI OEFFNEN
2260 GOSUB4150
2270 PRINT"<CLR>"
2280 PRINT
2290 PRINT
2300 PRINT"      RELATIVE DATEI OEFFNEN        "
2310 PRINT
2320 PRINT
2330 INPUT"DATEINAME";DN$
2340 PRINT
2350 I=0
2360 OPEN3,8,2,DN$+"-I,S,R"
2370 IF DS<>0 THEN PRINT DS$:GOTO2460
2380 FL=0
2390 INPUT#3,K
2400 PRINT K
2410 IF K>0 THEN FL=1
2420 IF I=MA THEN PRINT"DATEI VOLL":GOTO2460
2430 I=I+1
2440 INPUT#3,N$(I)
2450 IF ST=0 THEN 2420
2460 DCLOSE#3
2470 PRINT
2480 PRINT
2490 PRINT"          ZURUECK INS MENUE"
2500 PRINT"       BELIEBIGE TASTE DRUECKEN"
2510 GETKEY M$
2520 RETURN
2530 :
```

```
2540 :
2550 :
2560 REM RELATIVE DATEI ANLEGEN
2570 GOSUB 4150
2580 OPEN1,8,15,"I":CLOSE1
2590 PRINT"<CLR>"
2600 PRINT
2610 PRINT
2620 PRINT"        RELATIVE DATEI ANLEGEN          "
2630 PRINT
2640 PRINT
2650 INPUT"DATEINAME";DN$
2660 PRINT
2670 OPEN5,8,3,DN$:CLOSE5
2680 IF DS=0 THEN PRINT"BESTEHENDE ";:ELSE2750
2690 PRINT"DATEI UEBERSCHREIBEN ? (J/N)"
2700 PRINT
2710 GETKEY M$
2720 IF M$="J" THEN OPEN 2,8,2,"@:"+DN$+",L,"+CHR$(L) : GOTO 2780
2730 IF M$="N" THEN 2930
2740 GOTO 2710
2750 OPEN2,8,2,DN$+",L,"+CHR$(L)
2780 RECORD#2,MA,1
2790 IF DS=50 OR DS=0 THEN 2800:ELSE PRINT DS$:GOTO2880
2800 PRINT#2,CHR$(255)
2810 OPEN 3,8,3,DN$+"-I,S,W"
2820 PRINT#3,0
2830 FOR I=1 TO MA
2840 PRINT I;"<1xCRSR UP>"
2850 N$(I)=LEFT$(G$,LN)+STR$(I)
2860 PRINT#3,N$(I)
2870 NEXT I
2880 DCLOSE#3
2900 DCLOSE#2
2910 FL=1
2920 K=0
2930 PRINT
2940 PRINT
2950 PRINT"          ZURUECK INS MENUE"
2960 PRINT"        BELIEBIGE TASTE DRUECKEN"
2970 GETKEY M$
2980 RETURN
```

```
2990 :
3000 :
3010 :
3020 REM ADRESSENDATEI AUSDRUCKEN
3030 GOSUB 3970
3040 PRINT"<CLR>"
3050 PRINT
3060 PRINT
3070 PRINT"        ADRESSEN AUSDRUCKEN          "
3080 PRINT
3090 PRINT
3100 OPEN2,8,2,DN$
3110 OPEN4,4
3120 FORI=1TOK
3130 D=VAL(MID$(N$(I),LN+1,10))
3150 RECORD#2,D,1
3160 INPUT#2,R$
3170 PRINT#4,LEFT$(R$,LN);
3180 PRINT#4," ";
3190 PRINT#4,MID$(R$,LN+1,LV);
3200 PRINT#4," ";
3210 PRINT#4,MID$(R$,LN+LV+1,LS);
3220 PRINT#4," ";
3230 PRINT#4,MID$(R$,LN+LV+LS+1,LO);
3240 PRINT#4," ";
3250 PRINT#4,RIGHT$(R$,1)
3260 IF I-INT(I/64)*64=0 THEN BEGIN
3270 FOR J=1 TO 8
3280 PRINT#4
3290 NEXT J
3300 BEND
3310 NEXT I
3320 CLOSE4
3330 DCLOSE#2
3350 PRINT
3360 PRINT
3370 PRINT"          ZURUECK INS MENUE"
3380 PRINT"        BELIEBIGE TASTE DRUECKEN"
3390 GETKEY M$
3400 RETURN
3410 :
3420 :
```

```
3430 :
3440 REM ADRESSEN AUF ETIKETTEN DRUCKEN
3450 GOSUB 3970
3460 PRINT"<CLR>"
3470 PRINT
3480 PRINT
3490 PRINT" ADRESSEN AUF ETIKETTEN AUSDRUCKEN"
3500 PRINT
3510 PRINT
3520 OPEN2,8,2,DN$
3530 OPEN4,4
3540 FOR I=1 TO K
3550 D=VAL(MID$(N$(I),LN+1,10))
3570 RECORD#2,D,1
3580 INPUT#2,R$
3590 PRINT#4
3600 PRINT#4
3610 A=VAL(RIGHT$(R$,1))
3620 PRINT#4,AR$(A)
3630 PRINT#4,LEFT$(R$,LN+LV)
3640 PRINT#4,MID$(R$,LN+LV+1,LS)
3650 PRINT#4
3660 PRINT#4,MID$(R$,LN+LV+LS+1,LO)
3670 PRINT#4
3680 PRINT#4
3690 NEXT I
3700 CLOSE4
3710 DCLOSE#2
3730 PRINT
3740 PRINT
3750 PRINT"          ZURUECK INS MENUE"
3760 PRINT"        BELIEBIGE TASTE DRUECKEN"
3770 GETKEY M$
3780 RETURN
3790 :
3800 :
3810 :
3820 REM DATENSATZ LOESCHEN
3830 Z$=LEFT$(G$,LN)+MID$(N$(I),LN+1,10)
3840 FOR J=I+1 TO K
3850 N$(J-1)=N$(J)
3860 NEXT J
```

```
3870 N$(K)=Z$
3880 K=K-1
3890 PRINT"<8xCRSR DOWN>"
3900 PRINT"GELOESCHT"
3910 PRINT"<2xCRSR UP>"
3920 IF K=0 THEN FL=0
3930 RETURN
3940 :
3950 :
3960 :
3970 REM ABFRAGE OB DRUCKER EINGESCHALTET
3980 TRAP 4030
3990 OPEN4,4,0," "
4000 CLOSE4
4010 RETURN
4020 :
4030 IF ER<>5 THEN RESUME4010
4040 CLOSE4
4050 PRINT"<CLR>"
4060 PRINT
4070 PRINT
4080 PRINT "      BITTE DRUCKER EINSCHALTEN"
4090 PRINT "     UND BELIEBIGE TASTE DRUECKEN"
4100 GETKEY M$
4110 RESUME
4120 :
4130 :
4140 :
4150 REM ABFRAGE OB FLOPPY EINGESCHALTET
4160 TRAP 4210
4170 OPEN4,8,15,"I"
4180 CLOSE4
4190 RETURN
4200 :
4210 CLOSE4
4220 IF ER<>5 THEN RESUME 4190
4230 PRINT"<CLR>"
4240 PRINT
4250 PRINT
4260 PRINT "      BITTE FLOPPY EINSCHALTEN"
4270 PRINT "     UND BELIEBIGE TASTE DRUECKEN"
4280 GETKEY M$
```

```
4290 RESUME
4300 :
4310 :
4320 :
4330 REM PROGRAMM BEENDEN
4340 IF FL=0 THEN 4410
4350 OPEN3,8,3,"@:"+DN$+"-I,S,W"
4360 PRINT#3,K
4370 FOR I=1 TO MA
4380 PRINT#3,N$(I)
4390 NEXT
4400 DCLOSE#3
4410 END
```

Im weiteren Verlauf noch einige Anmerkungen zum Programmlisting, wobei wir uns nur auf diejenigen Punkte konzentrieren wollen, die sich von der sequentiellen Adressenverwaltung unterscheiden.

Zunächst gibt Zeile 130 in der Variablen L die Länge eines Datensatzes wieder, wobei bereits ein zusätzliches Zeichen für Carriage Return enthalten ist. Die maximale Anzahl der Datensätze wurde hier ebenfalls mit 200 angesetzt (Variable MA) und kann bei Bedarf bis auf 720 erhöht werden, was der maximalen Anzahl der Datensätze in einer relativen Datei entspricht.

Kommen wir nun zum Unterprogramm, das eine relative Datei anlegt (Zeile 2560). Zeile 2670 fragt ab, ob bereits eine Datei unter dem gleichen Namen DN$ angelegt ist. Dies geschieht durch kurz aufeinanderfolgendes Öffnen und Schließen der Datei und anschließender Fehlerabfrage. Ist die Nummer der Fehlermeldung in DS gleich Null, existiert eine solche Datei und der Bediener wird gefragt, ob diese Datei überschrieben werden soll. Anderenfalls entsteht eine Fehlermeldung mit der Nummer 62 (FILE NOT FOUND), worauf die Datei mit MA Records und der Recordlänge L angelegt wird (Zeile 2750 ff). Anschließend folgt ab Zeile 2810 die Anlage der sequentiellen Indexdatei, ebenfalls mit MA Elementen. Diese Indexdatei wird außerdem im Feld N$(I) angelegt, in das sie auch später zur Bearbeitung der Datei eingelesen wird und während der gesamten Bearbeitungsdauer dort verbleibt.

Auch bei der relativen Dateiverwaltung hätten wir anstelle der OPEN-Anweisung DOPEN verwenden können. Da dies jedoch in einigen Fällen problematisch werden kann, insbesondere wenn sich der Dateiname nicht aus einer Konstanten, sondern aus mehreren Elementen bzw. Variablen zusammensetzt, wollen wir es hier bei OPEN belassen. Falls Sie dennoch DOPEN verwenden möchten, müssen Sie sämtliche Variablen in Klammern setzen.

Somit kann eine relative Datei auf zweierlei Weise angelegt werden. Im folgenden Beispiel steht die Variable X$ für den Dateinamen und die Variable I für die Recordlänge:

```
OPEN 2,8,2,X$+",L,"+CHR$(I)
DOPEN#2, (X$) ,L(I)
```

Da die Indexdatei noch leer ist, wird als erster Wert die Zahl Null in sie geschrieben, gefolgt von MA Elementen, die aus 15 Sternchen und der fortlaufenden Recordnummer bestehen. Hier die ersten Elemente der Indexdatei:

```
 0
*************** 1
*************** 2
*************** 3
usw.
```

Die Sternchen bezeichnen einen leeren oder als gelöscht gekennzeichneten Record. Wird der Record später belegt, erscheint oben statt der Null die Anzahl der belegten Records, die ansonsten in der Variable K abgelegt ist und statt der Sternchen erscheint der betreffende Name als Suchindex. Da Namen nur in Strings gespeichert werden können, wandeln wir auch die entsprechenden Recordnummern mit der STR$-Anweisung in Strings um, damit wir Name und Recordnummer in einen String packen können. Wird die Recordnummer später benötigt, wird sie aus dem String herausgelöst und mit der VAL-Anweisung wieder in einen numerischen Wert umgewandelt.

Kommen wir nun zum "Öffnen" einer relativen Datei ab Zeile 2250. Hier wird lediglich die Indexdatei eingelesen, wobei davon ausgegangen wird, daß, wenn sich eine Indexdatei auf der Diskette befindet, auch die zugehö-

rige relative Datei vorhanden sein muß. Wird die Indexdatei nicht gefunden, erscheint die DOS-Fehlermeldung FILE NOT FOUND ERROR.

Der erste aus der Indexdatei eingelesene Wert gibt die Anzahl der bereits belegten Records wieder und wird sogleich der Variablen K zugeordnet. Die restlichen Elemente werden im Feld N$(I) abgelegt.

Im Unterprogramm "ADRESSEN SCHREIBEN/FORTSCHREIBEN" ab Zeile 610 wird dann mit dem K+1ten Datensatz fortgefahren. Ist K gleich Null, also die Datei leer, wird zur Eingabe des ersten Datensatzes aufgefordert. Nach beendeter Eingabe wird in Zeile 960 der Datensatz in den betreffenden Record geschrieben und anschließend in Zeile 990 die Indexdatei aktualisiert, indem der Name in sie eingetragen wird.

Im Menüpunkt "AENDERN/LOESCHEN" wird der gesuchte Name zunächst in der Indexdatei gesucht und dann der zugehörige Datensatz aus dem ebenfalls in der Indexdatei enthaltenen Record gelesen. Nach Durchführung der Änderung wird er dann wieder in dem gleichen Record abgelegt.

Zum Löschen wird das Unterprogramm ab Zeile 3820 aufgerufen. Dabei wird der Name in der Indexdatei mit Sternchen überschrieben, wobei allerdings die Recordnummer erhalten bleibt. Ist der zu löschende Datensatz an der Stelle I in der Indexdatei eingetragen, werden alle Elemente von I+1 bis K um eine Stelle nach unten aufgerückt und das Element des gelöschten Datensatzes an der Stelle K eingefügt. Da sich durch das Löschen die Anzahl der Datensätze um eins reduziert, erniedrigt sich abschließend der Wert von K ebenfalls um eins. Der gelöschte Record steht nun wieder für einen neuen Eintrag zur Verfügung.

Zum Sortieren der Indexeinträge dient das Unterprogramm ab Zeile 1940, wobei wieder das Sortierverfahren nach Shell zur Anwendung kommt. Meistens verschiebt sich erst durch das Sortieren, aber auch durch das Löschen, die Reihenfolge der Recordnummern im Verhältnis zu derjenigen im Indexverzeichnis.

Zu den Routinen, die die Adressen auf dem Drucker listen bzw. Etiketten ausdrucken, gibt es nur soviel zu sagen, daß dies in der Reihenfolge der Elemente in der Indexdatei geschieht, wobei der jeweilige Record eingelesen und anschließend ausgedruckt wird.

Wird bei Abschluß der Bearbeitung im Menü die Taste E gedrückt, verzweigt das Programm nach Zeile 4330, wo die Indexdatei in ihrem aktualisierten Zustand wieder auf die Diskette geschrieben wird und daraufhin das Programm endet.

7 Graphik und Sound

Im letzten Kapitel wollen wir uns noch mit den Graphik-, Farb- und Sound-Möglichkeiten auf dem 128 PC beschäftigen. Der 128 PC bietet, im Gegensatz zum Commodore 64, eine Vielzahl von BASIC-Anweisungen, die das Zeichnen von Farbgraphiken und die Erzeugung von Melodien auf einfache Weise ermöglichen. Somit wird die Graphik- und Soundprogrammierung auch für diejenigen zugänglich, die sich nicht mit der Registerbelegung der Bausteine auskennen. Beim Commodore 64 waren hierfür detaillierte Kenntnisse erforderlich, und man brauchte viel Geduld, um die unzähligen POKE- und PEEK-Befehle richtig einzugeben.

Um diesem Mißstand abzuhelfen, wurden von verschiedenen Firmen BASIC-Erweiterungen für den C-64 auf den Markt gebracht, die aber meist nicht mehr als ein Notbehelf waren. So brachte z.B. Commodore selbst den Super-Expander heraus, dessen zusätzliche BASIC-Anweisungen jetzt in das BASIC 7.0 des 128 PC übernommen wurden.

Wir wissen bereits, daß der 128 PC zwei Videoausgänge besitzt, nämlich einen Composite- und einen RGBI-Ausgang. Über den Composite-Ausgang können wir, ähnlich wie beim C-64, 40 Textzeichen pro Zeile und die normale Hires- und Mehrfarbengraphik ausgeben, was einer Auflösung von 200 x 320 Pixels (Bildpunkten) entspricht. Der RGBI-Ausgang dagegen arbeitet wahlweise auch im 80-Zeichen-Modus mit einer Auflösung von 200 x 640 Pixels.

Aber die Sache hat leider einen kleinen Haken, denn die Graphikbefehle gelten nur für Graphiken mit der geringeren Auflösung, die über den Composite-Ausgang ausgegeben werden. Im 80-Zeichen-Modus dagegen lassen sich nur 80 Zeichen, nicht aber hochauflösende Graphik mit 200 x 640 Pixels darstellen, jedenfalls nicht von BASIC aus. Es ist allerdings

möglich, beide Ausgänge gleichzeitig mit zwei Monitoren zu betreiben, wobei sich der RGBI-Monitor im 80-Zeichen-Textmodus und der Composite-Monitor im Graphik-Modus befindet. In dieser Konfiguration können Sie ein BASIC-Programm über den RGBI-Monitor eingeben, das Graphiken auf dem Composite-Monitor erscheinen läßt. Sie können also die Graphik auf dem Composite-Monitor betrachten, während das zugehörige BASIC-Programm auf dem RGBI-Monitor sichtbar bleibt.

Eine weitere Einschränkung besteht darin, daß der Composite-Monitor nicht im FAST-Modus betrieben werden kann. Dieser ermöglicht nur eine Ausgabe über den RGBI-Ausgang, über den ja von BASIC aus keine hochauflösende Graphik ausgegeben werden kann. Im FAST-Modus wird der 8502-Mikroprozessor mit einer Taktfrequenz von 2 MHz betrieben, während es im Normal- oder SLOW-Modus nur 1 MHz ist. Deshalb laufen im FAST-Modus die Programme auch doppelt so schnell ab. FAST und SLOW sind gleichzeitig BASIC-Anweisungen, die auf den jeweiligen Modus umschalten.

7.1 Bildschirmfarben und Graphikmodi

Der Bildschirm setzt sich grundsätzlich aus drei Bereichen zusammen: dem Vordergrund, dem Hintergrund und dem Rahmen. Sämtliche Bereiche können in 16 verschiedenen Farben erscheinen, von denen jede wiederum 8 verschiedene Helligkeitsstufen annehmen kann. Man spricht daher von Vordergrund-, Hintergrund- und Rahmenfarbe.

In den Rahmen können wir bekanntlich nichts hineinschreiben oder zeichnen, wohl aber in den inneren Bildschirm. Hierin ist die Hintergrundfarbe die Grundfarbe und die Vordergrundfarbe die Farbe, in der die Textzeichen oder Graphiken erscheinen.

Die Steuerung dieser Farben übernimmt die COLOR-Anweisung, die folgendermaßen definiert ist:

COLOR (Bereich), (Farbe)

Dabei gibt es folgende Bereiche:

Nummer	Bereich
0	Bildschirm-Hintergrund im 40-Zeichen- und Graphikmodus
1	Hires-Graphik-Vordergrund
2	Multicolor-Modus 1
3	Multicolor-Modus 2
4	Rahmenfarbe
5	Zeichenfarbe im 40/80-Zeichen-Textmodus
6	Bildschirm-Hintergrund im 80-Zeichen-Modus

Auf die Graphik-Bereiche 1 bis 3 kommen wir später noch zurück. Zunächst betrachten wir einmal die 16 Farben, die wir auf dem 128 PC erzeugen können:

Nummer	Farbe
1	Schwarz
2	Weiß
3	Rot
4	Cyan
5	Purpur
6	Grün
7	Blau
8	Gelb
9	Orange
10	Braun
11	Rosa
12	Dunkelgrau
13	Grau
14	Hellgrün
15	Hellblau
16	Hellgrau

Nun wollen wir ein Beispiel für die COLOR-Anweisung betrachten, in dem wir im normalen 40-Zeichen-Textmodus die Rahmenfarbe auf rot, die Hintergrundfarbe auf blau und die Vordergrund- (Zeichen-) Farbe auf gelb setzen:

```
10 COLOR 4, 3
20 COLOR 0, 7
30 COLOR 5, 8
```

Zur Auswahl der Zeichenfarbe können wir auch die Zifferntasten 1 bis 8 in der oberen Reihe verwenden. Werden diese Tasten zusammen mit der CTRL-Taste gedrückt, erscheinen die entsprechenden Zeichenfarben 1 bis 8; im Zusammenhang mit der C=-Taste sind es die Farben 9 bis 16.

GRAPHIC stellt eine weitere wichtige Anweisung dar, die wir zur Graphik-Programmierung benötigen. Mit GRAPHIC können nämlich sechs verschiedene Graphik-Modi ausgewählt werden. Die allgemeine Anweisungsform sieht so aus:

GRAPHIC (Modus), (Bildschirm löschen ja/nein)

Folgende Modi stehen zur Verfügung:

Modus	**Beschreibung**
0	Text 40 Zeichen pro Zeile und Blockgraphik
1	Hochauflösende Graphik
2	Hochauflösende Graphik mit Textfenster
3	Multicolor-Graphik
4	Multicolor-Graphik mit Textfenster
5	Text 80 Zeichen pro Zeile

Der zweite Parameter gibt an, ob der Bildschirm gelöscht werden soll oder nicht:

Parameter	**Bedeutung**
0	Bildschirm nicht löschen
1	Bildschirm löschen

Wenn wir nun in hochauflösende Graphik umschalten und gleichzeitig den Bildschirm löschen wollen, geben wir ein:

GRAPHIC 1,1

Beachten Sie dabei, daß der Bildschirm jetzt im Graphikmodus hängt und er somit sämtliche Eingaben über die Tastatur nicht mehr wiedergeben kann. Deshalb ist diese Anweisung nur im Programmodus zu empfehlen, wenn nach dem Zeichnen der Graphik der Text-Bilschirm infolge von

GRAPHIC 0 oder GRAPHIC 0,1

wieder eingeschaltet wird. Auf den zweiten Parameter kann man auch verzichten, wenn der Bildschirm nicht gelöscht werden soll. Er wird dann automatisch mit Null angenommen.

7.2 Text-/Blockgraphikmodus

Der Text-/Blockgraphikmodus steht Ihnen bereits unmittelbar nach dem Einschalten des Computers zur Verfügung und muß nicht erst gesondert angewählt werden. Sie haben sicher schon festgestellt, daß auf den meisten Tasten auch zwei Graphikzeichen abgebildet sind. Mit diesen Zeichen können Sie sogenannte Blockgraphiken erstellen und diese beliebig mit Text mischen. Wenn Sie die betreffende Taste zusammen mit der C=-Taste drücken, erscheint das linke, im Zusammenhang mit der Shift-Taste das rechte abgebildete Zeichen auf dem Bildschirm.

Dieser Graphikmodus heißt Blockgraphik, weil immer nur ein vorgegebener Block von 8x8 Pixels, aus denen sich jedes einzelne Graphikzeichen zusammensetzt, dargestellt werden kann. Diese Zeichen sind wie die übrigen Ziffern und Buchstaben fest vorgegeben und können nicht verändert werden. Sie eignen sich aber nur zur groben Darstellung von Abbildungen. Häufig verwendet man die verschiedenen Strichzeichen zur übersichtlichen Darstellung von Menüs oder sonstigen Bildschirmausgaben. Wir hätten sie auch in unserem Adressenverwaltungsprogramm (siehe Kapitel 6) verwenden können, um die einzelnen Menüpunkte zu umranden oder auf sonstige Weise besser hervorzuheben.

Im Text-/Blockgraphikmodus kann theoretisch jedes Zeichen eine andere Farbe erhalten, die wir entweder mit der COLOR-Anweisung oder mit den Farbtasten in der oberen Reihe vorgeben können. Wenn wir diese Tasten

allerdings im Programmodus verwenden, müssen wir sie genauso wie die Cursorsteuertasten oder die CLR-Taste programmieren. Dazu ist zunächst ein Anführungszeichen, dann die betreffende Taste zusammen mit der C=- bzw. CONTRL-Taste und schließlich wieder ein Anführungszeichen einzugeben. Dabei entstehen reverse Zeichen auf dem Bildschirm, die als Steuerzeichen für die entsprechende Farbe interpretiert werden.

7.3 Hochauflösende Graphik

Nun wollen wir uns mit der eigentlichen Graphik beschäftigen, zu der wir einen extra Graphikmodus in der GRAPHIC-Anweisung anwählen müssen. Die hochauflösende Graphik, auch Hires-Graphik genannt, ermöglicht es dem Programmierer, jeden einzelnen Bildpunkt (Pixel) gesondert anzusteuern. Da der Graphik-Bildschirm eine Auflösung von 200x320 Pixels besitzt, können Sie demgemäß insgesamt 64000 Bildpunkte setzen bzw. löschen. Die Bildschirm-Koordinaten erstrecken sich in der horizontalen Richtung von 0 bis 319 und in der vertikalen Richtung von 0 bis 199.

Eine Einschränkung müssen wir allerdings bei der hochauflösenden Graphik in Kauf nehmen. Wir können zwar 64000 Bildpunkte einzeln ansteuern, aber nur 64 Punkten zusammen einer Farbe zuordnen. Dies ist in der Größe des Farbspeichers begründet, der nur für die 1000 Bildschirmzeichen im Text- bzw. Blockgraphik-Modus ausgelegt ist. Wollten wir alle 64000 Punkte auch noch einzeln einfärben, brauchten wir einen riesigen Speicher, der den größten Teil des RAM-Speichers beanspruchen würde. In seinem jetzigen Zustand benötigt der Computer bereist 8k an Speicher für 64000 Pixels und nochmals 1k an Farbspeicher. Erhielte jedes Pixel seinen Farbspeicher, so benötigten wir insgesamt 72k.

Meistens verwendet man in der Hires-Graphik nur eine Hintergrund- und eine einheitliche Vordergrundfarbe, um eine einigermaßen brauchbare Farbtrennung zu gewährleisten. Für mehrfarbige Graphiken eignet sich der Multicolor-Modus, auf den wir noch zu sprechen kommen werden, weitaus besser.

Um Figuren im Graphikmodus zeichnen zu können, ist es nicht unbedingt erforderlich, jeden Bildpunkt einzeln zu setzen bzw. zu löschen, denn hierzu stellt uns BASIC 7.0 einige komfortable Anweisungen zur Verfügung.

Betrachten wir zunächst die DRAW-Anweisung, die Punkte oder Linien zeichnet bzw. löscht. Probieren Sie einmal folgendes Programm aus:

```
10 GRAPHIC 1,1
20 DRAW 1,160,100
30 GETKEY A$
40 GRAPHIC 0
```

Zeile 10 schaltet den Hires-Modus ein (1. Parameter) und sorgt dafür, daß der Graphikbildschirm gelöscht wird (2. Parameter). Zeile 20 setzt einen Punkt in die Bildschirmmitte (Koordinate 160/100). Da der erste Parameter eine 1 ist, wird er gesetzt, wäre er eine Null, würde der Punkt gelöscht werden. Wir könnten Zeile 20 auch so schreiben:

```
20 DRAW,160,100
```

Hier ist der erste Parameter nicht angegeben, wohl aber das Komma, das hinter ihm stehen muß. In diesem Fall wird der Wert 1 angenommen, d.h. der Punkt wird gesetzt.

Wenn wir in diesem Zusammenhang von Löschen sprechen, bedeutet das lediglich, daß die Graphik mit der Hintergrundfarbe gezeichnet und damit unsichtbar wird. Somit können wir auch sagen, daß der Parameter 1 die Vordergrundfarbe und der Parameter 0 die Hintergrundfarbe zum Zeichnen aktiviert.

Zeile 30 wartet dann schließlich auf einen Tastendruck, bevor Zeile 40 mit GRAPHIC 0 in den Textmodus zurückkehrt. Hätten wir die GETKEY-Anweisung ausgelassen, würde das Programm sofort nach Ausführung der Graphik-Anweisungen in den Textmodus zurückspringen und wir hätten kaum Zeit, das Ergebnis zu betrachten. Deshalb wollen wir auch in allen folgenden Beispielen die GETKEY-Anweisung beibehalten.

Mit diesem Beispiel sind die Möglichkeiten von DRAW aber noch nicht erschöpft. Nachfolgend ein Programm, das zwischen den Koordinatenpunkten 10/10 und 90/90 eine Linie zeichnet:

```
10 GRAPHIC 1,1
20 DRAW, 10,10 TO 90,90
30 GETKEY A$
40 GRAPHIC 0
```

Nun erweitern wir das Programm, indem wir die Zeile 25 einfügen:

```
10 GRAPHIC 1,1
20 DRAW, 10,10 TO 90,90
25 DRAW TO 80,110
30 GETKEY A$
40 GRAPHIC 0
```

Die neue Anweisung zeichnet von der Position, an der sich der Graphikcursor gerade befindet, eine Linie zur neuen Position 80/110.

Theoretisch könnten wir mit der DRAW-Anweisung alle Graphiken zeichnen, selbst wenn es nur punktweise wäre. Bei Kreisen müßten wir dann beispielsweise erst genügend viele Punkte des Kreisbogens berechnen und diese dann miteinander verbinden. Diese Mühe wird uns von BASIC 7.0 jedoch weitestgehend abgenommen.

Neben DRAW gibt es nämlich noch die beiden Anweisungen BOX und CIRCLE, die Rechtecke bzw. Kreise zeichnen. Betrachten wir zunächst einmal BOX:

```
10 GRAPHIC 1,1
20 BOX,50,50,100,100
30 GETKEY A$
40 GRAPHIC 0
```

Dieses Programm zeichnet uns ein Rechteck, genaugenommen ein Quadrat, bei dem zwei diagonal gegenüberliegende Eckpunkte vorgegeben sein müssen. In unserem Fall haben sie die Koordinaten 50/50 und 100/100. Der erste Parameter, für den wir nur das Komma setzen, ist auch hier gleichbedeutend mit dem in der DRAW-Anweisung.

Wir können die Rechteckfigur auch drehen, indem wir einen weiteren Parameter als Drehwinkel in Grad angeben:

```
10 GRAPHIC 1,1
20 BOX,50,50,100,100,45
30 GETKEY A$
40 GRAPHIC 0
```

Mit diesem Programm entsteht das gleiche Quadrat wie im vorigen Beispiel; es ist nur um einen Winkel von 45 Grad verdreht. Wollen wir es zusätzlich ausfüllen, müssen wir noch einen weiteren Parameter hinter die Winkelangabe setzen. Im folgenden Beispiel wollen wir dies einmal ausprobieren:

```
10 GRAPHIC 1,1
20 BOX,50,50,100,100,45,1
30 GETKEY A$
40 GRAPHIC 0
```

Die 1 als letzter Parameter gibt die Anweisung, das Quadrat auszufüllen. Wünschen wir keine Drehung, lassen wir die Winkelangabe fort und setzen nur das abschließende Komma. Dies gilt übrigens für alle Parameter in Graphik-Anweisungen, wenn sie nicht benutzt werden.

Bevor wir die BOX-Anweisung verlassen, abschließend noch ein Programm, das durch wiederholtes Drehen eines Quadrates das Muster einer Rosette erzeugt:

```
10 REM ROSETTE
20 GRAPHIC 1,1
30 FOR I=0 TO 8
40 BOX,90,30,230,170,I*10
50 NEXT
60 GETKEY A$
70 GRAPHIC 0,0
```

Wir sehen an diesem Beispiel, daß die Parameter auch in Form arithmetischer Ausdrücke angegeben werden können.

Als nächstes kommen wir nun zur CIRCLE-Anweisung, die in erster Linie Kreise und Ellipsen zeichnet. Neben den Mittelpunktkoordinaten muß dabei zumindest noch der Radius angegeben werden. Mit dem folgenden Programm können Sie vier Kreise ineinander zeichnen:

```
10 REM KREISE
20 GRAPHIC 1,0
30 CIRCLE, 160,100,100
40 CIRCLE, 160,100,80
50 CIRCLE, 160,100,60
60 CIRCLE, 160,100,40
70 CHAR,17,12,"HURRA!"
80 GETKEY A$
90 GRAPHIC 0
```

Sämtliche Kreise haben ihren Mittelpunkt in der Bildschirmmitte (160/100). Der äußerste Kreis hat einen Radius von 100, der sich beim zweiten Kreis auf 80, beim dritten auf 60 und beim vierten auf 40 reduziert. Die CHAR-Anweisung in Zeile 70 dient zur Beschriftung von Graphiken und setzt das Wort "HURRA!" in die Kreismitte. Die angegebenen Parameter 17/12 geben die Anfangsposition der Textausgabe an, allerdings nicht in Pixels, sondern als normale Zeichenposition wie im Textmodus.

Im folgenden Beispiel wollen wir eine Ellipse zeichnen. Dafür geben wir noch einen zweiten Radius als Parameter an:

```
10 GRAPHIC 1,1
20 CIRCLE,160,100,100,50
30 GETKEY A$
40 GRAPHIC 0
```

Neben den beiden ersten Parametern, die den Mittelpunkt bestimmen, erscheint zunächst der Radius in horizontaler und anschließend der Radius in vertikaler Richtung. Die CIRCLE-Anweisung bietet aber noch weitaus mehr Möglichkeiten. Zur besseren Veranschaulichung und zum klareren Verständnis finden Sie nachfolgend sämtliche Parameter aufgeführt:

```
CIRCLE   Vorder-/Hintergrundfarbe,
         Mittelpunktkoordinate X-Achse,
         Mittelpunktkoordinate Y-Achse,
         Radius X-Achse,
         Radius Y-Achse,
         Winkel in Grad am Kreisbogenanfang (Standard = 0),
         Winkel in Grad am Kreisbogenende (Standard = 360),
         Drehung im Uhrzeigersinn zusätzlich (Standard = 0),
         Winkel zwischen zwei Segmenten (Standard 2 Grad)
```

Unter Ausnutzung dieser Parameter können wir auch Vielecke zeichnen. So erzeugt die Anweisung

```
CIRCLE,160,100,50,,,,,120
```

ein Dreieck und

```
CIRCLE,160,100,70,,,,,45
```

ein Achteck. Hier noch ein Programm, das eine gelbe Spirale auf dunkelblauem Grund zeichnet:

```
10 REM SPIRALE
20 GRAPHIC 1,1
25 COLOR 0,15 : COLOR 1,8
30 A = 180 : REM RADIUSAENDERUNGEN PRO UMDREHUNG
40 B = 6   : REM ANZAHL UMDREHUNGEN
50 R = 100 : REM GROESSTER RADIUS
60 C = A*B
70 FOR I=1 TO C
80 CIRCLE,160,100,(I-1)/C*R,,0,B*360/C,B*(I-1)/C*360
90 NEXT
100 GETKEY A$
110 GRAPHIC 0
```

Das Programm läßt die Spirale vom Mittelpunkt aus entstehen, wobei es 180 Mal pro Umdrehung den Radius neu berechnet und daraufhin mit der CIRCLE-Anweisung ein Kreissegment von 2 Grad zeichnet.

Jetzt wollen wir eine Möglichkeit betrachten, Figuren nicht nur zu zeichnen, sondern auch farblich auszufüllen. Die BOX-Anweisung läßt dies bereits für Rechtecke zu, aber auch beliebige andere Flächen können mit Hilfe der PAINT-Anweisung ausgefüllt werden. Als Beispiel dafür soll uns ein Kreis dienen:

```
10 GRAPHIC 1,1
20 CIRCLE, 160,100,50
30 PAINT,160,100
40 GETKEY A$
50 GRAPHIC 0
```

Dieses Programm zeichnet einen Kreis, der anschließend mit Hilfe der PAINT-Anweisung farblich ausgefüllt wird. PAINT benötigt neben dem Komma als Ersatz für den Ein-/Aus-Parameter noch eine X- und eine Y-Koordinate für die Stelle, an der mit dem Ausfüllen begonnen werden soll. In unserem Fall ist es die Kreismitte. Wichtig ist lediglich, daß die auszufüllende Fläche vollständig geschlossen ist und die Koordinaten der PAINT-Anweisung innerhalb dieser Fläche liegen. Wir hätten somit auch jeden anderen Punkt innerhalb des Kreises auswählen können.

7.4 Multicolor-Modus

Hires-Graphiken ermöglichen zwar eine Auflösung von 320x200 Pixels, haben aber meist eine mäßige Farbauflösung zur Folge, besonders wenn es darum geht, mehr als zwei Farben einzusetzen. Die Farben lassen sich an den Stoßstellen nicht sauber trennen und verwaschen ineinander. Darüber hinaus können immer nur 64 Pixels in einem 8x8 Pixel großen Kästchen eine gemeinsame Farbe annehmen. Aus diesen Gründen arbeiten die meisten Hires-Programme ohnehin nur mit der Hintergrundfarbe und einer einzigen Vordergrundfarbe.

Im Multicolor-Modus erstellte Graphiken lassen dagegen in einem 8x8 Pixel großen Kästchen insgesamt vier Farben zu: die Hintergrund- und die Vordergrundfarbe sowie zwei weitere Farben. Dieser Vorteil muß allerdings mit einem Nachteil erkauft werden, denn die Auflösung der Graphik nimmt auf 200x160 Pixels ab. Dabei stehen in der X-Richtung mit 160 Pixels nur halb so viele Bildpunkte zur Verfügung wie im Hires-Modus. Dies hat andererseits den Vorteil, daß die Farbtrennung bei weniger Punkten besser funktioniert als bei vielen.

Vom Multicolor-Modus gibt es zwei Variationen, nämlich den Multicolor-Modus 1 und 2. Angewählt werden diese beiden Arten durch die COLOR-Anweisung, wobei der Multicolor-Modus 1 mit COLOR 2 und der Multicolor-Modus 2 mit COLOR 3 angewählt wird.

Der Multicolor-Modus 1 arbeitet in der bereits besprochenen Weise, während der Modus 2 noch eine zusätzliche Eigenschaft besitzt, die es in keinem anderen Bereich bzw. Graphikmodus gibt. Sämtliche mit COLOR 3 erstellten Graphikpassagen ändern nämlich schlagartig ihre Farbe, sobald eine neue COLOR 3-Anweisung gegeben wird. Dadurch entsteht in gewisser Weise ein Leuchtreklame-Effekt. So können Sie z.B. mit COLOR 3,8

mehrere gelbe Sterne auf den Bildschirm zeichnen, die dann bei COLOR 3,3 plötzlich rot oder bei COLOR 3,7 plötzlich blau werden.

7.5 Graphik und Text

Wir kennen bereits die CHAR-Anweisung, mit der Graphiken beschriftet werden können. So schreibt z.B. die Anweisung

CHAR,17,12,"HURRA!"

das Wort "HURRA!" an die Stelle, die im Textmodus die Position 17,12 einnehmen würde, obwohl wir uns gerade im Graphikmodus befinden. Achten Sie bei Anwendung der CHAR-Anweisung darauf, daß sich im Textmodus die horizontalen Koordinaten von 0 bis 39 und die vertikalen von 0 bis 25 erstrecken (40-Zeichen-Bildschirm), während wir für die Hires-Graphik 320x200 und für die Multicolor-Graphik 320x160 Bildpunkte zur Verfügung haben.

Eine weitere Möglichkeit zur Unterbringung von Text bietet die Bildschirmaufteilung in zwei Bereiche, wobei die oberen 20 (Text-) Zeilen für Graphik und die unteren 5 Zeilen für Text reserviert sind. In das Textfenster können mit PRINT beliebige Zeichen ausgegeben werden. Diese Bildschirmaufteilung gibt es für Hires- und für Multicolor-Graphiken; sie wird mit GRAPHIC 2 bzw. GRAPHIC 4 erzeugt.

7.6 Sprites

In den vorangehenden Abschnitten haben wir gelernt, Graphiken im Hires- oder Multicolor-Modus zu erstellen. Stellen Sie sich einmal vor, Sie entwerfen einen Ballon mit einer Gondel, den Sie auf- oder absteigen lassen möchten. Sie müssen dafür nicht nur eine Graphik entwerfen, sondern diese darüber hinaus auch noch bewegen.

Natürlich besteht die Möglichkeit, dies mit den Anweisungen zu tun, die wir bisher kennengelernt haben, denn wir können eine Figur zeichnen und wieder löschen (erster Parameter in den einzelnen Graphik-Anweisungen). Nun könnten wir sie um einige Pixels verschieben und wieder neu zeichnen, wieder löschen und verschieben usw. Wiederholt sich dieser Vorgang schnell genug, entsteht der Eindruck, daß sich das Objekt bewegt.

Wenn wir ein derartiges Programm zu schreiben versuchen, werden wir bald feststellen, daß es sehr umfangreich und kompliziert wird. Außerdem wird es uns kaum möglich sein, die Objekte in schneller Folge aufzubauen und wieder zu löschen. Der BASIC-Interpreter braucht nämlich auch seine Zeit, die unzähligen Anweisungen auszuführen, die für einen solchen Vorgang nötig sind.

Zur schnellen Bewegung von Objekten auf dem Bildschirm bietet nur der Commodore 128 PC die Sprite-Graphik an. BASIC 7.0 enthält einige sehr nützliche Befehle zur Aufstellung und Bewegung von Sprites. Dabei sind Sprites für Commodore-Computer nichts neues. Der C-64 unterstützte bereits die Sprite-Graphik, selbst wenn sie dort nur auf umständliche Weise durch zahlreiche POKE- und PEEK-Befehle oder Maschinenprogramme zu realisieren war.

Ein Sprite ist eine rechteckige Fläche mit einer Breite von 24 und einer Höhe von 21 Pixels. Auf dem Commodore 128 können Sie bis zu acht Sprites erzeugen und gleichzeitig über den Bildschirm ziehen lassen. Sprites sind übrigens wichtige Programmierwerkzeuge und zwar aus folgenden Gründen:

Jedes Sprite hat seine eigene Form und Farbe und bewegt sich auf seiner eigenen Bildschirmebene. Diese Ebenen existieren völlig unabhängig voneinander, wobei es auch keine Rolle spielt, von welcher Beschaffenheit die einzelnen Sprites sind. Wenn Sie nun ein Sprite über den Bildschirm bewegen, hat dies keinerlei Einfluß auf die übrige Bildschirmgraphik oder die anderen Sprites. Durch diese komfortable Einrichtung brauchen Sie sich beim Programmieren keine Gedanken zu machen, ob ein Sprite durch ein anderes überlagert wird oder nicht. Müßten Sie sich hierüber den Kopf zerbrechen, wären Ihre Programme mit Sicherheit um einiges komplizierter.

Als weiteres bewegen sich Sprites zu jedem vorgegebenen Punkt auf dem Bildschirm. Sie müssen nur die gewünschten Koordinaten eingeben, der Rest geschieht dann automatisch. Sie können auch einen Winkel und die Geschwindigkeit vorgeben, wonach sich das Sprite bewegen soll. Selbst wenn es an den Bildschirmrand stößt, muß dies kein Hindernis sein, denn es kann am anderen Ende des Bildschirms wieder hervorkommen. Diese hervorragenden Eigenschaften lassen sich alle mit einem Minimum an Programmieraufwand nutzen. Dies ist auch der Grund, weshalb es für den C-64 so viel gute Graphik-Software gibt.

Es besteht auch die Möglichkeit, mehrere Sprites zu einer großen Figur zusammenzusetzen. So ergeben vier Sprites ein großes Gebilde mit 96x84 Pixels, was schon einen erheblichen Teil der Bildschirmfläche ausmacht. Auch können Sie jedem Sprite eine Priorität zuordnen, wodurch es sich entweder vor oder hinter anderen Bildschirmdaten bewegen kann. Dadurch werden dreidimensionale Effekte erzielt, die beispielsweise den Eindruck vermitteln, daß sich ein Ungeheuer hinter einem Berg versteckt, und von dort plötzlich hervortritt. Selbst die Kollision eines Sprites mit einem anderen Sprite oder mit der Hintergrundgraphik kann vorprogrammiert werden, so daß das Programm in eine vorgegebene Routine springt, wenn eine Kollision auftritt.

Damit Sprites auf dem Bildschirm erscheinen, ist es nicht unbedingt erforderlich, den Hires- oder Multicolor-Modus zu aktivieren. Sie können auch auf dem normalen Textbildschirm sichtbar sein, während Sie ansonsten Programme eingeben oder ausführen. Die Sprites bleiben davon vollkommen unberührt. Selbst wenn Sie ein BASIC-Programm neu mit RUN starten oder mit NEW löschen, hat das keinerlei Einfluß auf einmal definierte Sprites. Sie stehen solange zur Verfügung, wie der Computer eingeschaltet bleibt.

Um Sprites zu erstellen, gibt es zwei Möglichkeiten, die wir hier kurz betrachten wollen:

Sie zeichnen das Sprite-Muster mit normalen Graphik-Anweisungen wie DRAW, CIRCLE oder BOX auf den Bildschirm und legen es in einer Stringvariablen ab. Dies wird durch die SSHAPE-Anweisung möglich. So legt beispielsweise

```
SSHAPE A$,80,80,103,100
```

das definierte Spritemuster in der Variablen A$ ab, wobei 80,80 und 103,100 die diagonal gegenüberliegenden Eckpunkte darstellen. Dabei ist auf die richtige Größe des Sprites von 21x24 Pixels zu achten. SSHAPE wird übrigens allgemein zum Abspeichern von Graphiken verwendet, die dann mit GSHAPE wieder gesetzt werden können. Somit würde unsere Graphik mit

```
GSHAPE A$,20,20
```

in die Bildschirmposition 20,20 gesetzt werden, wobei die angegebenen Koordinaten den linken oberen Eckpunkt angeben.

Dies nur zur allgemeinen Information. Wir wollen jedoch unsere Graphik im Augenblick nicht mit GSHAPE wiedergeben, sondern sie einem der acht möglichen Sprites zuordnen. Hierzu wählen wir das erste Sprite aus:

SPRSAV 1, A$

Jetzt ist das Sprite Nr. 1 definiert, so daß wir es über den Bildschirm ziehen lassen können. Doch bevor wir dies tun, sehen wir uns noch eine andere Möglichkeit an.

BASIC 7.0 besitzt einen eingebauten komfortablen Sprite-Editor, mit dem es ein leichtes ist, Sprites zu erzeugen. Geben Sie zunächst einmal

SPRDEF

ein. In der linken oberen Bildschirmecke erscheint ein großes Kästchen, das ungefähr den halben Bildschirm beansprucht. Darunter steht der Text SPRITE NUMBER?, der Sie auffordert, eine Nummer zwischen 1 und 8 einzugeben. Wir wählen hier wieder die Nummer 1. Sollte unter dieser Nummer bereits ein Sprite belegt sein, erscheint es in stark vergrößerter Form in dem Kästchen und in normaler Größe rechts oben im Bildschirm. Wir können es, falls wir es nicht mehr benötigen, mit der SHIFT/CLR-Taste löschen.

Im linken oberen Eckpunkt erscheint ein Kreuz (Pluszeichen) als Cursor, der auch auf die normalen Cursorsteuertasten anspricht. Mit den Ziffern-tasten 2 bis 4 können wir jetzt einzelne Punkte setzen, während die Zif-ferntaste 1 Punkte löscht. Dabei geben wir das Spritemuster in achtfacher Vergrößerung ein, wobei ein Kästchen von 8x8 Pixels in Wirklichkeit nur einem Pixel entspricht. Rechts auf dem Bildschirm erscheint das gleiche Muster noch einmal in Originalgröße.

Auf diese Weise ist die Erzeugung jedes beliebigen Musters leicht möglich. Nach Abschluß der Eingabe drücken wir SHIFT/RETURN und können dann entweder ein weiteres Sprite definieren oder nochmals die RETURN-Taste drücken, um den Vorgang abzuschließen.

Jetzt ist das Sprite fertig definiert und verwendungsbereit. Es besteht je-doch auch die Möglichkeit, es in einer Stringvariablen abzulegen, wozu wiederum die SPRSAV-Anweisung dient, diesmal allerdings mit vertausch-ten Parametern. Die Anweisung

SPRSAV C$,1

ordnet das binäre Pixelmuster von Sprite 1 der Stringvariablen C$ zu.

Nun wollen wir unser Sprite hervorholen; dabei spielt es keine Rolle, in welchem Graphikmodus wir uns gerade befinden. Die Anweisung

SPRITE 1,1

läßt das Sprite rechts oben auf dem Bildschirm erscheinen und zwar an der Stelle, wo es während der Definition in seiner Originalgröße vorhanden war. Soll das Sprite wieder verschwinden, geben wir

SPRITE 1,0

ein. Da die SPRITE-Anweisung noch mehrere Optionen bietet, wollen wir sie zunächst in ihrer allgemeinen Form betrachten:

SPRITE Nummer, Ein/Aus, Farbe, Priorität,
X-Ausdehnung, Y-Ausdehnung, Modus

Die Nummer entspricht der Spritenummer zwischen 1 und 8; für Ein/Aus steht eine 1, wenn das Sprite sichtbar, und eine Null, wenn es unsichtbar sein soll. Für die Farbe ist einer der 16 Farbcodes zu setzen. Die Priorität ist gleich 0, wenn sich das Sprite vor den anderen Bildschirmdaten bewegt und 1, wenn es von ihnen verdeckt wird. Mit den nächsten Parametern können wir das Sprite sowohl in X-, als auch in Y-Richtung in seiner Größe verdoppeln. In diesem Fall sind entweder einer oder beide Parameter auf 1 zu setzen, während sie für die Normalgröße auf Null bleiben. Der letzte Parameter gibt den Modus wieder, wobei 1 dem Hires- und Null dem Multicolor-Modus entspricht.

Wenn wir eingeben

SPRITE 5, 1, 14, 1, 1, 1

wird demgemäß das 5. Sprite sichtbar; es erhält die dunkelblaue Farbe, bewegt sich hinter den anderen Bildschirminformationen und wird zusätzlich sowohl in X- als auch in Y-Richtung auf die doppelte Ausdehnung vergrößert.

Mit den MOVSPR-Anweisung können wir das Sprite beliebig auf dem Bildschirm hin- und herbewegen.

MOVSPR 1, 40, 30

setzt das Sprite 1 an die Stelle 40/30. Werden die Koordinaten mit einem Vorzeichen versehen, wie z.B. in +50, -30, bewegt sich das Sprite von seinem gegenwärtigen Standpunkt um 50 Pixels nach rechts und um 30 Pixels nach oben. Bei

MOVSPR 1,45#4

gerät das Sprite 1 ständig in Bewegung. Es bewegt sich in einem Winkel von 45 Grad im Uhrzeigersinn mit der Geschwindigkeitsstufe 4. Läuft es über den Bildschirmrand hinaus, erscheint es wieder auf der entgegengesetzten Seite und läuft weiter. Insgesamt gibt es 16 Geschwindigkeitsstufen (0-15), wobei 0 der langsamsten und 15 der schnellsten Stufe entspricht.

Wird ein solches Sprite mit einer SPRITE-Anweisung gestoppt, werden seine gegenwärtigen Koordinaten und Bewegungsdaten registriert, so daß es von derselben Stelle aus neu gestartet und in der gleichen Weise weiterlaufen kann.

Abschließend wollen wir noch einen kurzen Blick auf die Behandlung von Spritekollisionen werfen. Tritt ein solches Ereignis (Event) auf, verzweigt das Programm in ein Unterprogramm (Trapping) zur weiteren Behandlung dieses aufgetretenen Ereignisses. Deshalb spricht man in diesem Zusammenhang auch von Event Trapping.

Zum Abfragen und Abfangen von solchen Zusammenstöße dient die COLLISION-Anweisung. Ihr folgt die Nummer des Ereignisses und dann die Nummer der Zeile, in die im Falle dieses Ereignisses verzweigt werden soll. Dabei bedeutet

0	Kollision zweier Sprites
1	Kollision zwischen Sprite und Bildschirmdaten
2	Kollision infolge Lightpen (Lichtgriffel)

Somit verzweigt die Anweisung

COLLISION 0, 1000

in Zeile 1000, falls zwei Sprites zusammengestoßen sind. Sämtliche angesteuerten Routinen müssen mit einer RETURN-Anweisung abschließen, da sie wie normale Unterprogramme behandelt werden. Nach Ausführung von RETURN fährt das Programm mit der nächsten Anweisung fort.

Die COLLISION-Anweisung ist in etwa mit der TRAP-Anweisung zu vergleichen, die im Falle eines auftretenden Fehlers eine besondere Routine anspringt. Dabei spielt es keine Rolle, welche BASIC-Anweisung während des Fehlers bzw. der Kollision gerade ausgeführt wird, denn das Programm springt sofort in die betreffende Routine und setzt nach deren Abarbeitung seine Ausführung an der auslösenden Stelle fort. Diese Art der Programmunterbrechung wird übrigens oft bei der Maschinenprogrammierung angewandt. Hier nennt man solche Programmunterbrechungen Interrupts.

Die COLLISION-Anweisung wird außer Kraft gesetzt, wenn sie ohne Zeilennummer geschrieben wird, z.B.:

```
COLLISION 1
```

Da die COLLISION-Anweisung lediglich angibt, daß eine Kollision zwischen zwei Sprites oder einem Sprite und den Bildschirmdaten stattgefunden hat, gibt es darüber hinaus noch die BUMP-Funktion, die nähere Angaben zur Art des Zusammenstoßes liefert. Sie muß in der Routine stehen, die infolge von COLLISION aufgerufen wird.

```
A = BUMP (0)
```

gibt die Sprites an, die miteinander kollidiert sind. Dabei hat der Wert in Klammern die gleiche Funktion wie bei der COLLISION-Anweisung.

```
B = BUMP (1)
```

ermittelt somit die Sprites, die mit den Bildschirmdaten, d.h. mit normalen Graphiken oder Textzeichen, zusammengestoßen sind.

Das durch BUMP erzeugte Ergebnis, das in unserem Beispiel den Variablen A bzw. B zugeordnet wird, ist als 8-Bit-Wert binär zu interpretieren und liegt im Bereich zwischen 0 und 255 dezimal bzw. 0000 0000 und 1111 1111 binär. Dabei entspricht jedes Bit einem der acht möglichen Sprites. Ist z.B. das rechte Bit auf 1 gesetzt, so ist Sprite 1 von der Kollision betroffen, beim zweiten Bit von rechts ist es Sprite 2 usw. Dabei können auch zwei Bit gleichzeitig gesetzt sein, wie im Falle von BUMP (0), da ja hier der

Zusammenstoß zwischen zwei Sprites abgefragt wird. Dagegen ist es bei BUMP (1) immer nur ein Bit. Enthält z.B. die Variable A in unserem ersten Beispiel den Wert 18, so entspricht dieser der Binärzahl 0001 0010, was bedeutet, daß Sprite 2 mit Sprite 5 zusammengestoßen ist.

Abschließend werfen wir noch einen Blick auf zwei weitere Funktionen, die Spriteparameter abfragen und wiedergeben. Die RSPPOS-Funktion ermittelt Position und Geschwindigkeit eines Sprites. Ihre allgemeine Form lautet:

RSPPOS (Spritenummer, Parameter-Wert)

Für den Parameter können verschiedene Werte vorgegeben werden:

Wert	ermittelt Parameter
0	derzeitige X-Position
1	derzeitige Y-Position
2	Geschwindigkeitsstufe (0-15)

So ermittelt die Eingabe

A = RSPPOS (3,2)

die augenblickliche Geschwindigkeit von Sprite 3 und legt das Ergebnis in der Variablen A ab.

Die Funktion

RSPRITE (Spritenummer, Attribut-Nr.)

liefert verschiedene Spriteattribute, die nachfolgend aufgeführt sind:

Attribut-Nr.	liefert als Ergebnis
0	1, wenn Sprite aktiviert, anderenfalls 0
1	Spritefarbe 1 - 16
2	1, wenn Sprite Priorität über Hintergrund hat, anderenfalls 0
3	1, wenn Sprite in X-Richtung gedehnt, anderenfalls 0

4	1, wenn Sprite in Y-Richtung gedehnt, anderenfalls 0
5	1, wenn Multicolor-Modus aktiviert, anderenfalls 0

So ermittelt die Eingabe

A = RSPRITE (2, 1)

die Farbe von Sprite 2, dessen Nummer in A abgelegt wird.

7.7 Musik und Sound

Mit BASIC 7.0 können Sie nicht nur Graphik und Sprites, sondern auch Melodien und Geräusche auf einfache Weise programmieren. Zwar war dies bereits beim C-64 generell möglich, jedoch wie bei der Graphik-Programmierung nur mit POKE- und PEEK-Befehlen zu erreichen. In diesem Abschnitt wollen wir nun die komfortablen BASIC-Soundanweisungen näher betrachten.

Bevor wir damit beginnen, achten Sie darauf, daß an Ihren Computer auch ein Lautsprecher angeschlossen ist. Falls Sie ein Fernsehgerät als Bildschirm benutzen, genügt es, dazu den Lautstärkeregler aufzudrehen, da der Ton auf diese Weise automatisch übertragen wird. Haben Sie dagegen einen Schwarzweiß- oder Farbmonitor angeschlossen, prüfen Sie zunächst, ob dieser auch mit einem Lautsprecher ausgestattet ist, was bei vielen Monitoren nicht der Fall ist.

Hat Ihr Monitor keinen eingebauten Lautsprecher, müssen Sie sich anderweitig behelfen. Entweder benutzen Sie für die Soundprogrammierung ein Fernsehgerät oder Sie schließen zusätzlich einen Verstärker mit Lautsprecher oder eine Stereoanlage an, die ohnehin eine viel bessere Klangwiedergabe erzeugt. Sollten Sie gleichzeitig Monitor und Verstärker bzw. Stereoanlage an den Audio-/Videoausgang anschließen, benötigen Sie ein Verteilerstück, das Sie entweder kaufen oder selbst herstellen müssen. Besitzen Sie dagegen einen RGBI-Monitor, können Sie diesen an den RGBI-Ausgang und den Verstärker bzw. die Stereoanlage an den Composite- oder Audio-/Video-Ausgang anschließen.

Betrachten wir zunächst einmal die PLAY-Anweisung, mit der wir am einfachsten Melodien erzeugen können. Da der 128 PC bereits mit den Hüllkurven 10 verschiedener Musikinstrumente vorprogrammiert ist, können wir sofort damit beginnen, eine Melodie zu programmieren und abzuspielen. Starten wir unseren ersten Versuch mit der Tonleiter in C-Dur:

```
PLAY "C D E F G A B"
```

Auf das hohe C mußten wir dabei vorläufig verzichten, da es bereits zur nächst höheren Oktave gehört. Sie sehen aber, daß prinzipiell nur die einzelnen Noten in der Form einzugeben sind, in der man sie auch ausspricht. Die einzige Ausnahme bildet die Note H, die im Amerikanischen - wie auch bei der Soundprogrammierung - mit B bezeichnet wird.

Bevor wir fortfahren, werfen wir noch einen Blick auf die einzelnen zusätzlichen Parameter für die PLAY-Anweisung.

Der 128 PC kann Töne in ingesamt acht Oktaven wiedergeben, die mit O0 bis O8 bezeichnet werden. Jetzt wollen wir unsere Tonleiter nochmals spielen, diesmal allerdings über drei volle Oktaven:

```
PLAY "O2 CDEFGAB O3 CDEFGAB O4 CDEFGAB O5 C"
```

Die Bezeichnungen O2 bis O5 geben hierbei die jeweilige Oktave vor (Buchstabe O, nicht Ziffer Null).

Als nächstes wählen wir unter verschiedenen Instrumenten aus, die mit dem T-Parameter vorgegeben werden:

Nummer	Instrument
0	Klavier
1	Akkordeon
2	Zirkusorgel
3	Trommel
4	Flöte
5	Gitarre
6	Cembalo
7	Orgel
8	Trompete
9	Xylophon

Die folgenden Töne wollen wir mit einer Flöte in der 4. Oktave spielen:

PLAY "T4 O4 CDEC"

Der U-Parameter gibt verschiedene Lautstärken vor, die mit U0 (leise) bis U9 (laut) anzugeben sind. Darüberhinaus besteht auch die Möglichkeit, Zwischentöne zu erzeugen, wobei den einzelnen Noten ein #-Zeichen (ein halber Ton höher) oder ein $-Zeichen (ein halber Ton tiefer) voranzustellen ist (z.B. #C oder $G).

Mit der PLAY-Anweisung können wir die Dauer eines Tones beeinflussen. Hier die Parameter, die den einzelnen Noten vorangestellt sein müssen:

W	ganze Note
H	halbe Note
Q	Viertelnote
I	Achtelnote
S	sechszentel Note
.	eineinhalbfache Länge wie angegeben (punktierte Note)

Stehen diese Werte vor einem R, dann beziehen sie sich auf eine Pause, in der kein Ton erklingt. Die folgende PLAY-Anweisung spielt die ersten Noten der Melodie "Oh du lieber Augustin" mit einem Akkordeon:

PLAY "T1 O4 HG QAGFEC HC"

Bei dieser Anweisung steht T1 für Akkordeon und O4 für die 4. Oktave. Der erste Ton G ist eine halbe Note, die Töne A, G, F, E, C sind Viertelnoten und das letzte C ist wieder eine halbe Note.

Auf dem 128 PC können Sie auch zwei- und dreistimmige Melodien spielen. Im folgenden Beispiel erzeugen wir einen Akkord in der 4. Oktave:

PLAY "O4 V1 C V2 E V3 G"

Dabei spielt die erste Stimme V1 ein C, die zweite Stimme V2 ein E und die dritte Stimme V3 ein E. Obwohl die einzelnen Stimmen hintereinander aufgeführt sind, erscheinen sie dennoch (fast) gleichzeitig. Bei mehrstimmigen Melodien ist zusätzlich für jede Note immer die zugehörige Stimme anzugeben.

Die Klangform der einzelnen Instrumente ist in der jeweils zugehörigen Hüllkurve gespeichert. Falls Sie mit einem anderen Instrument spielen oder den Klang eines vorgegebenen Instruments abändern möchten, können Sie dies mit der ENVELOPE-Anweisung tun, die folgende Parameter enthält, die jeweils durch Kommas getrennt sein müssen:

ENVELOPE	Hüllkurvennummer	(0 bis 9),
	Anschwellzeit	(0 bis 15),
	Abschwellzeit	(0 bis 15),
	Beharrungszeit	(0 bis 15),
	Ausklingzeit	(0 bis 15),
	Wellenform:	(0 Dreieck)
		(1 Sägezahn)
		(2 Rechteck)
		(3 Rauschen)
		(4 Ringmodulation)
	Impulsbreite	(0 bis 4095)

Neben der PLAY-Anweisung, die vorwiegend zum Spielen von Melodien eingesetzt wird, gibt es noch die SOUND-Anweisung. Sie dient zur Erzeugung besonderer Soundeffekte mit veränderlichem Frequenzgang oder sonstigen außergewöhnlichen Eigenschaften. Die SOUND-Anweisung ist folgendermaßen definiert, wobei jeder Parameter durch Komma abgetrennt sein muß:

SOUND	Stimme	(1, 2 oder 3),
	Frequenzwert	(0 bis 65535),
	Dauer	(0 bis 32767 sechzigstel Sek.)
	Klangeffekt	(0 anschwellend)
		(1 abschwellend)
		(2 oszillierend)
	Maximalfrequenz für anschwellende Klangeffekte	(0 bis 65535)
	Dauer der Anschwellung	(0 bis 32767)
	Wellenform	(0 Dreieck)
		(1 Sägezahn)
		(2 Rechteck)
		(3 Rauschen)
	Impulsbreite	(0 bis 4095)

Der Frequenzparameter errechnet sich nach folgender Formel:

Parameterwert = Frequenz (Hz) / 0.06097

Nachfolgend eine Tabelle mit den wichtigsten Noten, ihrer Frequenz und dem zugehörigen Parameterwert. Die Parameterwerte verdoppeln sich für die nächsthöhere bzw. halbieren sich für die nächstniedrigere Oktave.

Note	Frequenz (Hz)	Parameterwert
C	262	4297
C#	277	4543
D	294	4822
D#	311	5100
E	330	5412
F	349	5724
F#	370	6069
G	392	6429
G#	415	6807
A	440	7217
A#	466	7643
B	494	8102
C	523	8578

Zur Regelung der Lautstärke eines durch SOUND erzeugten Tons dient die VOL-Anweisung mit Werten von 0 bis 15. VOL 0 erzeugt dabei einen leisen und VOL 15 einen lauten Ton.

Anhang

Verzeichnis der BASIC-Anweisungen und -Funktionen

ABS	Bestimmt Absolutwert einer Zahl
AND	Logische UND-Verknüpfung
APPEND	Öffnet seq. Datei zum Anfügen von Daten
ASC	Erzeugt ASCII-Wert eines Zeichens
ATN	Arcus-Tangens-Funktion
AUTO	Automatische Zeilennumerierung
BACKUP	Kopiert gesamten Disketteninhalt
BANK	Wählt Speicherbank aus
BEGIN...BEND	Faßt in IF...THEN...ELSE mehere Zeilen zusammen
BLOAD	Lädt Maschinenprogramm in Speicher
BOOT	Lädt und startet CP/M von Diskette
BOX	Zeichnet Rechtecke
BSAVE	Speichert beliebigen Speicherbereich auf Floppy
BUMP	Liefert Sprite-Nummer nach Sprite-Kollision
CATALOG	Gibt Inhaltsverzeichnis der Diskette aus
CHAR	Beschriftet Hires-Graphiken
CHR$	Erzeugt Zeichen aus ASCII-Codes
CIRCLE	Zeichnet Kreise
CLOSE	Schließt logisches File
CLR	Löscht sämtliche Variablen
CMD	Lenkt Bildschirmausgabe auf Peripheriegerät um
COLLECT	Löscht offene Dateien und reorganisiert Diskette
COLLISION	Fragt Sprite-Kollision ab
COLOR	Setzt Farbe für Graphik und Text
CONCAT	Verbindet zwei seq. Dateien miteinander
CONT	Programmfortführung nach Unterbrechung
COPY	Kopiert Disketten-Datei
COS	Cosinus-Funktion
DATA	Speichert Datenwerte für READ-Anweisung

DCLEAR	Schließt alle Kanäle zur Diskettenstation
DCLOSE	Schließt Kanal zur Diskettenstation
DEC	Bestimmt Dezimalwert einer hexadezimalen Zahl
DEF FN	Definiert Funktion
DELETE	Löscht Programmzeilen
DIM	Dimensioniert Felder
DIRECTORY	Liest Inhaltsverzeichnis von Diskette
DLOAD	Lädt Programm von Diskette
DO...LOOP	Programm-Endlosschleife
DOPEN	Öffnet Kanal zur Diskettenstation
DRAW	Zeichnet Punkte und Linien
DS	Liest Fehlernummer des Diskettenlaufwerks
DS$	Gibt Floppy-Fehlermeldung aus
DSAVE	Schreibt Programm auf Diskette
DVERIFY	Überprüft Programmabspeicherung auf Diskette
EL	Liefert Zeilennummer bei Auftreten eines Fehlers
ELSE	Alternative bei IF...THEN
END	Ende eines BASIC-Programms
ENVELOPE	Definiert Hüllkurve für Soundprogrammierung
ER	Liefert Fehlercode
ERR$ (ER)	Liest Fehlermeldung
EXIT	Verläßt DO...LOOP-Schleife
EXP	Errechnet Potenz zur Zahl e
FAST	Schaltet auf FAST-Modus um
FETCH	Holt Daten aus beliebiger Speicherbank
FILTER	Setzt Klangfilter-Parameter
FN	Ruft durch DEF FN definierte Funktion auf
FOR...TO..STEP	Leitet Zählschleife ein
FRE	Bestimmt freien BASIC-Speicher
GET	Holt einzelnes Zeichen
GET#	Holt einzelnes Zeichen von Peripheriegerät
GETKEY	Wartet, bis Taste gedrückt
GO64	Schaltet in C-64-Modus um
GOSUB	Ruft Unterprogramm auf

GOTO	Springt in angegebene Programmzeile
GRAPHIC	Wählt Graphik-Modus aus
GSHAPE	Liest Graphik aus String
HEADER	Formatiert Disketten
HELP	Listet fehlerhafte Programmzeile
HEX$	Formt Dezimalzahl in Hexadezimalstring um
IF..THEN..ELSE	Fragt Bedingung ab
INPUT	Holt Eingabewert vom Bildschirm
INPUT#	Holt Daten von Peripheriegerät
INSTR	Gibt Position eines Strings in einem anderen an
INT	Bestimmt ganzzahligen Teil einer Zahl
JOY	Fragt Joystickposition ab
KEY	Belegt Funktionstasten
LEFT$	Gibt linken Teil eines Strings wieder
LEN	Bestimmt Stringlänge
LET	Zuordnungsanweisung
LIST	Listet Programm
LOAD	Lädt Programm von Floppy oder Kassette
LOCATE	Setzt Graphikcursor
LOG	Berechnet natürlichen Logarithmus
MID$	Gibt Teil eines Strings wieder
MONITOR	Ruft Maschinensprache-Monitor auf
MOVSPR	Bewegt Sprite über den Bildschirm
NEW	Löscht BASIC-Programm im Speicher
NEXT	Ende einer Zählschleife
NOT	Logische NICHT-Verknüpfung
ON	Verzweigt aufgrund eines Zahlenwertes
OPEN	Öffnet logisches File
PAINT	Füllt geschlossene Fläche einer Graphik aus
PEEK	Liest Inhalt einer Speicherzelle
PEN	Fragt Lightpen ab
PLAY	Spielt in Strings abgelegte Musiknoten
POINTER	Liefert Adresse einer Variablen im Speicher
POKE	Schreibt Wert in Speicherzelle

POS	Gibt Cursorposition in Zeile an
POT	Fragt Paddles ab
PRINT	Gibt Daten auf Bildschirm aus
PRINT#	Gibt Daten auf Peripheriegerät aus
PRINT USING	Formatierte Ausgabe von Zahlen und Strings
PUDEF	Definiert Steuerzeichen für PRINT USING
RCLR	Liefert Farbcode für Text und Graphik
READ	Liest DATA-Werte und legt sie in Variablen ab
RECORD	Setzt Schreib-/Lesezeiger für relative Dateien
REM	Beginn einer Kommentarzeile
RENAME	Benennt Diskettendateien um
RENUMBER	Neunumerierung des BASIC-Programms
RESTORE	Setzt DATA-Zeiger auf beliebige Zeilennummer
RESUME	Rückkehr aus einer Fehlerbehandlungsroutine
RETURN	Rücksprung aus einem Unterprogramm
RGR	Liefert Nummer des eingestellten Graphik-Modus
RIGHT$	Gibt rechten Teil eines Strings wieder
RND	Erzeugt Zufallszahl
RRG	Weist Variablen die Werte der Prozessorregister zu
RSPPOS	Liefert Position und Geschwindigkeit eines Sprites
RSPRCOLOR	Liefert akt. Code des Multicolor-Modus für Sprites
RSPRITE	Liefert Spriteattribute
RUN	Startet Programmablauf
RWINDOW	Liest Parameter des eingestellten Windows
SAVE	Speichert Programm auf Diskette oder Kassette
SCALE	Verändert Maßstab bei Hires-Graphik
SCNCLR	Löscht Text- oder Graphikbildschirm
SCRATCH	Löscht Diskettendatei
SGN	Bestimmt Vorzeichen einer Zahl
SIN	Sinusfunktion
SLEEP	Hält Programmausführung für die angegebene Zeit an
SLOW	Schaltet auf SLOW-Modus zurück
SOUND	Erzeugt beliebige Toneffekte
SPC	Dient zur Formatierung des Bildschirms

SPRCOLOR	Setzt Farben für Sprites im Multicolor-Modus
SPRDEF	Ruft Sprite-Editor auf
SPRITE	Setzt Sprite-Attribute
SPRSAV	Speichert Sprite in String oder umgekehrt
SQR	Berechnet Quadratwurzel
SSHAPE	Speichert Graphik in String ab
STASH	Überträgt Daten in Speicherbank
STATUS (ST)	Fragt Status ab
STEP	Schrittweite in FOR...NEXT-Schleife
STOP	Unterbricht BASIC-Programm
STR$	Wandelt Zahl in String um
SWAP	Tauscht Daten zwischen Speicherbänken aus
SYS	Ruft Maschinenprogramm auf
TAB	Setzt Tabulator auf Bildschirm
TAN	Tangens-Funktion
TEMPO	Gibt Spielgeschwindigkeit für PLAY vor
TI$	Abfrage der internen Uhr (HH,MM,SS)
TIME (TI)	Abfrage der internen Uhr (1/16stel Sekunden)
TRAP	Verzweigt bei Fehler in Fehlerbehandlungsroutine
TROFF	Schaltet Ablaufverfolgung aus
TRON	Schaltet Ablaufverfolgung ein
UNTIL	Setzt Bedingung für DO...LOOP
USR	Übermittelt Variable an Maschinenprogramm
VAL	Wandelt Ziffernstring in Zahl um
VERIFY	Prüft, ob Programm richtig abgespeichert (nach SAVE)
VOL	Setzt Lautstärke für SOUND-Anweisung
WAIT	Wartet auf bestimmten Speicherzelleninhalt
WHILE	Setzt Bedingung für DO...LOOP
WIDTH	Setzt Strichstärke für Graphik
WINDOW	Definiert Bildschirmfenster
XOR	Logische EXKLUSIV-ODER-Verknüpfung

ASC- und CHR$-Code-Tabelle

0		20	<DEL>	40	(	60	<
1		21		41	)	61	=
2		22		42	*	62	>
3		23		43	+	63	?
4		24		44	,	64	@
5	<WHT>	25		45	-	65	A
6		26		46	.	66	B
7		27	<ESCAPE>	47	/	67	C
8	<SH C= DISA>	28	<RED>	48	0	68	D
9	<SH C= ENAB>	29	<CRSR RIGHT>	49	1	69	E
10		30	<GRN>	50	2	70	F
11		31	<BLU>	51	3	71	G
12		32	<SPACE>	52	4	72	H
13	<RETURN>	33	!	53	5	73	I
14	<LOWER CASE>	34	"	54	6	74	J
15		35	#	55	7	75	K
16		36	$	56	8	76	L
17	<CRSR DOWN>	37	%	57	9	77	M
18	<RVS ON>	38	&	58	:	78	N
19	<HOME>	39	'	59	;	79	O

ASC- und CHR$-Code-Tabelle (Fortsetzung)

80	P	100		120		140	
81	Q	101		121		141	<SH RETURN>
82	R	102		122		142	<UPPER CASE>
83	S	103		123		143	
84	T	104		124		144	<BLK>
85	U	105		125		145	<CRSR UP>
86	V	106		126		146	<RVS OFF>
87	W	107		127		147	<CLR>
88	X	108		128		148	<INST>
89	Y	109		129		149	
90	Z	110		130	<FLASH ON>	150	
91	[	111		131	<FLASH OFF>	151	
92	£	112		132		152	
93	]	113		133		153	
94	↑	114		134		154	
95	←	115		135		155	
96		116		136		156	<PUR>
97		117		137		157	<CRSR LEFT>
98		118		138		158	<YEL>
99		119		139		159	<CYN>

ASC- und CHR$-Code-Tabelle (Fortsetzung)

160	<SH SPACE>	168	▄	176	▗	184	▔
161	▌	169	◤	177	┴	185	▁
162	▄	170	▐	178	┬	186	▕
163	▔	171	├	179	┤	187	▖
164	▁	172	▗	180	▏	188	▝
165	▏	173	└	181	▍	189	┘
166	▒	174	┐	182	▐	190	▘
167	▕	175	▂	183	▔	191	▚

Codes 192 - 223 identisch mit 96 - 127
Codes 224 - 254 identisch mit 160 - 190
Code 255 identisch mit 126

Verzeichnis der Fehlermeldungen

Nummer ER	Fehlermeldung ERR$ (ER)
1	TOO MANY FILES
2	FILE OPEN
3	FILE NOT OPEN
4	FILE NOT FOUND
5	DEVICE NOT PRESENT
6	NOT INPUT FILE
7	NOT OUTPUT FILE
8	MISSING FILE NAME
9	ILLEGAL DEVICE NUMBER
10	NEXT WITHOUT FOR
11	SYNTAX
12	RETURN WITHOUT GOSUB
13	OUT OF DATA
14	ILLEGAL QUANTITY
15	OVERFLOW
16	OUT OF MEMORY
17	UNDEF'D STATEMENT
18	BAD SUBSCRIPT
19	REDIM'D ARRAY
20	DIVISION BY ZERO
21	ILLEGAL DIRECT
22	TYPE MISMATCH
23	STRING TOO LONG
24	FILE DATA
25	FORMULA TOO COMPLEX
26	CAN'T CONTINUE
27	UNDEF'D FUNCTION
28	VERIFY
29	LOAD

30	BREAK
31	CAN'T RESUME
32	LOOP NOT FOUND
33	LOOP WITHOUT DO
34	DIRECT MODE ONLY
35	NO GRAPHICS AREA
36	BAD DISC
37	BEND NOT FOUND
38	-LINE NUMBER TOO LARGE
39	-UNRESOLVED REFERENCE
40	UNIMPLEMENTED COMMAND
41	FILE READ

Floppy-DOS-Fehlermeldungen

Nummer (DS)	Fehlermeldung (DS$)
0	OK (kein Fehler)
1	FILES SCRATCHED
2 - 19	ohne Bedeutung
20	READ ERROR (Blockheader nicht gefunden)
21	READ ERROR (SYNCH-Zeichen nicht gefunden)
22	READ ERROR (kein Datenblock vorhanden)
23	READ ERROR (Prüfsummenfehler im Datenblock)
24	READ ERROR (Hardwarefehler)
25	WRITE ERROR
26	WRITE PROTECT ON
27	READ ERROR (Prüfsummenfehler im Header)
28	WRITE ERROR (zu langer Datenblock)
29	DISC ID MISMATCH
30	SYNTAX ERROR (allgemein)
31	SYNTAX ERROR (ungültiger Befehl)
32	SYNTAX ERROR (zu langer Befehlscode)
33	SYNTAX ERROR (ungültiger Dateiname)
34	SYNTAX ERROR (Kein Dateiname angegeben)
39	SYNTAX ERROR (ungültiger Befehl)
50	RECORD NOT PRESENT
51	OVERFLOW IN RECORD
52	FILE TOO LARGE
60	WRITE FILE OPEN
61	FILE NOT OPEN
62	FILE NOT FOUND
63	FILE EXISTS
64	FILE TYPE MISMATCH
65	NO BLOCK
66	ILLEGAL TRACK AND SECTOR
67	ILLEGAL SYSTEM T OR S

70	NO CHANNEL
71	DIRECTORY ERROR
72	DISC FULL
73	DOS MISMATCH
74	DRIVE NOT READY

Index

Spitzen-Software für Commodore 128/128 D

WordStar 3.0 mit MailMerge

Der Bestseller unter den Textverarbeitungsprogrammen für PCs bietet Ihnen bildschirmorientierte Formatierung, deutschen Zeichensatz und DIN-Tastatur sowie integrierte Hilfstexte. Mit MailMerge können Sie Serienbriefe mit persönlicher Anrede an eine beliebige Anzahl von Adressen schreiben und auch die Adreßaufkleber drucken.

WordStar/MailMerge für den Commodore 128 PC
Bestell-Nr. MS 103 (5¼"-Diskette)

Hardware-Anforderungen: Commodore 128 PC, Diskettenlaufwerk, 80-Zeichen-Monitor, beliebiger Commodore-Drucker oder ein Drucker mit Centronics-Schnittstelle

Für nur DM 199,–* (sFr. 178,–/öS 1890,–*)

***inkl. MwSt. Unverbindliche Preisempfehlung**



MicroPro
WordStar 3.0
mit MailMerge für den Commodore 128 PC

5¼"-Diskette
im Floppy 1541-Format

Und dazu die weiterführende Literatur:



Mit diesem Buch haben Sie eine wertvolle Ergänzung zum WordStar-Handbuch: Anhand vieler Beispiele steigen Sie mühelos in die Praxis der Textverarbeitung mit **WordStar** ein. Angefangen beim einfachen Brief bis hin zur umfangreichen Manuskripterstellung zeigt Ihnen dieses Buch auch, wie Sie mit Hilfe von MailMerge Serienbriefe an eine beliebige Anzahl von Adressen mit persönlicher Anrede senden können.

Best.-Nr. 90181
ISBN 3-89090-181-6
DM 49,– (sFr. 45,10/öS 382,20)

Erhältlich bei Ihrem Buchhändler.

Markt&Technik-Produkte erhalten Sie in den Fachabteilungen der Warenhäuser, im Versandhandel, in Computerfachgeschäften oder bei Ihrem Buchhändler.

701321



Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0



JÜRGEN HÜCKSTÄDT

Jahrgang 1949, ist Diplomingenieur für Wasserbau. Im Rahmen seiner langjährigen Tätigkeit in einem Ingenieurbüro kam er schon frühzeitig mit Personal Computern in Berührung. Er entwickelte verschiedene Softwarepakete für wasserwirtschaftliche und geodätische Anwendungen und wurde schließlich mit der Leitung der EDV-Abteilung beauftragt. Seit einigen Jahren leitet er Seminare für BASIC- und Assemblerprogrammierung und ist darüber hinaus auch als Übersetzer, Lektor und Autor von EDV-Fachliteratur tätig.

BASIC 7.0 auf dem Commodore 128

Ganz gleich, ob Sie bereits über Programmierkenntnisse verfügen oder nicht, dieses Buch wird Ihnen helfen, den größtmöglichen Nutzen aus dem leistungsstarken BASIC 7.0 des Commodore 128 PC zu ziehen. Selbst wenn Sie sich noch nie mit Computern befaßt haben, steht Ihnen dieses Buch zur Seite, um die BASIC-Programmierung von den einfachsten Grundzügen an zu erlernen. Als unermüdlicher Helfer begleitet es Sie Schritt für Schritt durch alle Lernphasen, die hervorragenden Eigenschaften des BASIC 7.0 zunehmend besser zu verstehen und anzuwenden.

Auf den einfachsten Grundlagen aufbauend eignen Sie sich bei der Durcharbeitung dieses Buches alle notwendigen Kenntnisse an, um immer anspruchsvollere Aufgabenstellungen zu bewältigen.

Aus dem Inhalt:
- Programmschleifen und -verzweigungen
- Listenverarbeitung
- Indexsequentielle Dateiverwaltung
- Sortierverfahren
- Grafik- und Sonderprogrammierung

Ein unentbehrliches Lehrbuch und Nachschlagewerk für jeden Commodore 128 PC-Besitzer!

ISB N 3-89090-149-2



Markt&Technik

DM 52,-
sFr. 47,80
öS 405,60